От издателей русского перевода

На мировом рынке компьютерной литературы существует множество книг, предназначенных для обучения основным алгоритмам и используемых при программировании. Их довольно много, и они в значительной степени конкурируют между собой. Однако среди них есть особая книга. Это трехтомник "Искусство программирования" Д. Э. Кнута, который стоит вне всякой конкуренции, входит в золотой фонд мировой литературы по информатике и является настольной книгой практически для всех, кто связан с программированием.

Мы как издатели видим ценность книги в том, что она предназначена не столько для обучения технике программирования, сколько для обучения, если это возможно, "искусству" программирования, предлагает массу рецептов усовершенствования программ и, что самое главное, учит самостоятельно находить эти рецепты.

Ни для кого не секрет, что наши программисты являются одними из наиболее высококвалифицированных специалистов в мире. Они достойно представляют за рубежом отечественную школу программирования и информатики, которая внесла значительный вклад в формирование фундаментальных основ компьютерных наук. Для сохранения такого уровня и продвижения вперед необходимо своевременное издание на русском языке книг, отражающих основные мировые достижения в этой области. Трехтомник "Искусство программирования" Д. Э. Кнута — одна из таких книг.

Мы гордимся тем, что библиотеки программистов, преподавателей, студентов, старшеклассников и многих других пополнятся этой классической книгой и что тем самым мы внесем свой вклад в формирование более глубокого понимания основ компьютерных наук. Мы глубоко убеждены, что книга "Искусство программирования" Д. Э. Кнута способна приблизить человека к совершенству. Надеемся, наше издание на русском языке этой замечательной книги еще раз подтвердит, что истинные ценности с годами не устаревают.

– Виктор Штонда, Геннадий Петриковец, Алексей Орлович, издатели

О книге "Искусство программирования"

У каждой книги своя судьба. Одни появляются незаметно и так же незаметно исчезают в потоке времени, покрываясь пылью на полках библиотек. Другие в определенный период пользуются спросом у узкого круга специалистов, пока им на смену не приходят новые справочники. Третьи, поднимаясь над временем, оказывают мощное влияние на технологическое развитие общества. Книг, относящихся к последней категории, не так уж и много. Их выход в свет — всегда праздник. Проходят годы, изменяются технологии, но новые поколения с постоянным интересом перечитывают их страницы. Именно к таким книгам относится предлагаемый читателю многотомный труд известного американского ученого Дональда Эрвина Кнута "Искусство программирования".

Прошло почти 30 лет со времени первого издания в 1972 году в США этой книги. Она была переведена на большинство языков мира, в том числе и на русский. К настоящему времени на территории стран СНГ трехтомник Д. Э. Кнута стал библиографической редкостью. В 1998 году в США вышло третье издание "Искусства программирования". В нем сохранена последовательность изложения материала прежних версий, но значительно расширен список ссылок, в который включены свежие и наиболее важные результаты, добавлены новые упражнения и комментарии, устранены неточности. Учитывая популярность во всем мире "Искусства программирования", давно следовало ожидать появления нового переводного издания на русском языке, которое вы и держите в руках.

В чем же успех "Искусства программирования" Д. Э. Кнута?

Во-первых, эта книга — великолепное учебное пособие по составлению и анализу компьютерных алгоритмов. Ее разделы могут быть включены во многие университетские курсы по технологиям программирования, теории алгоритмов, дискретной математике. Книгу могут изучать и школьники старших классов, знакомые с основами программирования. В качестве основного языка записи алгоритмов автор выбрал язык машинных команд гипотетического универсального компьютера MIX. Это позволяет строить оптимальные программы с учетом особенностей вычислительных машин. Перенести MIX-программы на реальные ЭВМ или переписать их на языках высокого уровня не составляет особого труда. Логика работы программ почти всегда поясняется простыми блок-схемами.

Во-вторых, тщательно подобранный материал, вошедший в книгу, включает в себя основные фундаментальные классы алгоритмов, которые в том или ином виде наиболее часто встречаются в практике программирования.

В-третьих, немаловажным фактором успеха книги Д. Э. Кнута является энциклопедичность изложения. Профессор Кнут отличается уникальной способностью отслеживать проблему от исторических предпосылок ее зарождения до современного состояния. Многочисленные ссылки на работы старых мастеров (вплоть до времен античности), заключенные в современный контекст, создают у читателя особое чувство причастности к историческому развитию научных идей и методов.

В-четвертых, следует отметить мастерство изложения. Книга рассчитана на широкий круг читателей — от начинающих студентов до программистов-профессионалов. Каждому будет интересно изучать компьютерные алгоритмы на своем уровне. Материал

самодостаточен. Для понимания сути методов не требуется знания особых разделов математики или специальных технологий программирования. Прослеживается определенная "музыкальная" композиция сюжетного построения (дома у Д. Э. Кнута есть небольшой орган, на котором он играет).

Список составляющих успеха "Искусства программирования" можно легко продолжить.

Автор этих строк прослушал курс "Искусство программирования" в изложении профессора Кнута в 1976–1977 годах во время стажировки в Станфордском университете. Тогда формировалась алгоритмическая основа технологий программирования, у истоков которой стоял Д. Э. Кнут. Было много обсуждений, семинаров, творческих замыслов.

Значительные книги всегда связаны с судьбой автора. Дональд Эрвин Кнут начал работу над "Искусством программирования" в 1962 году. Продолжает ее и сейчас. У него много планов. Впереди новые тома "Искусства программирования", которых с нетерпением ждут читатели.

— Профессор Анатолий Анисимов

От редактора перевода

Со времени первого издания книги "Искусство программирования" Д. Э. Кнута прошло около 25 лет. Тем не менее книга не только не устарела, но по-прежнему остается основным руководством по искусству программирования, книгой, по которой учатся понимать суть и особенности этого искусства.

За эти годы на английском языке вышло уже третье издание 1-го и 2-го томов, а также второе издание 3-го тома. Автор внес в них значительные изменения и существенные дополнения. Достаточно сказать, что число упражнений практически удвоилось, а многие упражнения, включенные в предыдущие издания (особенно ответы к ним), модифицированы. Существенно дополнены и переделаны многие главы и разделы, исправлены неточности и опечатки, добавлены многочисленные новые ссылки на литературу, использованы теоретические результаты последних лет.

Значительно преобразилась глава 3, особенно разделы 3.5 и 3.6, а также разделы 4.5.2, 4.7, 5.1.4, 5.3, 5.4.9, 6.2.2, 6.4, 6.5 и др.

Естественно, возникла необходимость в новом издании книги.

Перевод выполнен по третьему изданию 1-го и 2-го томов и второму изданию 3-го тома. Кроме того, учтены дополнения и исправления, любезно предоставленные автором.

При переводе мы старались сохранить стиль автора, обозначения и манеру изложения материала. В большинстве случаев использовались термины, принятые в научной литературе на русском языке. При необходимости приводились английские эквиваленты. По многим причинам, в частности из-за сложности некоторых разделов, читать книгу "Искусство программирования" далеко непросто. Одной из причин, которые затрудняют понимание книги, является манера изложения автора; привыкнув к ней, можно существенно облегчить чтение.

Из-за обилия материала (часто мало связанного между собой) невозможно построить книгу так, чтобы различные понятия и определения вводились сразу же при первом упоминании о них. Поэтому в главе 1 могут обсуждаться без ссылок понятия, строгие определения которых приводятся в 3-м томе. Именно поэтому так велика роль предметного указателя, без которого понимание книги было бы существенно затруднено. Надеемся, что читатель не будет удивлен, найдя ссылки на главы 7, 8 и последующие не вошедшие в предлагаемые три тома главы. Мы вместе с автором надеемся, что очень скоро они будут опубликованы и, безусловно, сразу же появятся в русском переводе в качестве продолжения этого издания.

Следует также обратить внимание на далеко не всегда стандартные обозначения, которыми пользуется автор. Так же, как и определения, эти обозначения могут появиться в 1-м томе, а вводиться во 2-м. Поэтому без указателя обозначений пользоваться книгой было бы чрезвычайно трудно. Хочу также обратить внимание на запись `[A]`, где `A` — некоторое высказывание. Эта запись встречается в формулах, а иногда и в тексте, и обозначает величину, равную индикатору `A`.

— Профессор Ю. В. Козаченко

ПРЕДИСЛОВИЕ

Уважаемые читатели!
Вы держите в руках книгу,
издать которую Вы просили нас в тысячах писем.
Нам пришлось потратить годы на то,
чтобы самым тщательным образом проверить
и перепроверить бесконечное множество рецептов
и отобрать для вас самые лучшие, самые интересные,
самые совершенные.
Теперь без тени сомнения мы можем сказать,
что если вы будете следовать инструкциям,
то каждое блюдо будет получаться у вас таким же, как и у нас,
даже если раньше вы никогда не занимались
приготовлением пищи.

— *Поваренная книга Мак-Колла* (1963)

Процесс подготовки программ для цифрового компьютера — это очень увлекательное занятие. И дело не только в том, что оно оправдывает себя с экономической и научной точек зрения; оно может вызвать также эстетические переживания, подобные тем, которые испытывают творческие личности при написании музыки или стихов. Вы держите в руках первый том многотомного издания, цель которого — дать читателю разнообразные знания и умения, из которых и состоит ремесло программиста.

Последующие главы *не* являются введением в компьютерное программирование; предполагается, что вы уже имеете некоторый опыт в этой области. На самом деле предъявляемые к читателю требования очень просты; тем не менее начинающему программисту потребуются время и практика, чтобы понять, что собой представляет цифровой компьютер. Итак, читатель должен иметь

a) некоторое представление о том, как работает цифровой компьютер с хранимой программой; при этом необязательно разбираться в электронике, главное — понимать, каким образом команды можно сохранять в памяти компьютера, и затем последовательно их выполнять;

b) способность ставить задачу с помощью четких и определенных терминов, понятных компьютеру (у компьютеров нет разума, присущего человеку, поэтому они делают в точности то, что им приказывают, не больше и не меньше; именно этот факт обычно труднее всего уяснить начинающим пользователям);

c) знание самых простых компьютерных методов, таких как организация циклов (повторное выполнение некоторого набора команд), а также использование подпрограмм и переменных с индексами;

d) знание распространенных компьютерных терминов, например "память", "регистры", "биты", "плавающая точка", "переполнение", "программное обеспечение"; большинство терминов, которые не определены в тексте, поясняются в алфавитном указателе в конце каждого тома.

Эти четыре условия, вероятно, можно объединить в одном требовании: читатель должен иметь опыт написания и отладки по меньшей мере четырех программ хотя бы для одного компьютера.

Я старался писать эти книги так, чтобы они могли служить нескольким различным целям. Во-первых, они представляют собой справочное пособие, в котором сосредоточены знания из нескольких важных областей науки. Во-вторых, они могут использоваться в качестве пособий для самообразования и учебников по программированию или информатике для университетов. В связи с этим я включил в текст большое количество упражнений и предоставил ответы на большинство из них. Кроме того, я попытался сосредоточить внимание на фактах, вместо того чтобы "лить воду" и заниматься общими рассуждениями.

Этот трехтомник предназначен для всех, кто серьезно интересуется компьютерами, а не только для профессионалов. В сущности, одна из моих главных целей состояла в том, чтобы сделать методы программирования более доступными для специалистов из других областей. Как правило, эти специалисты получают большие преимущества, используя компьютеры, но не могут позволить себе тратить время на поиски необходимой информации, крупицы которой разбросаны по множеству технических журналов.

Тему этих книг можно сформулировать следующим образом: "Нечисленный анализ". Компьютеры обычно ассоциируются с решением численных задач, таких как нахождение корней уравнения, численное интерполирование, интегрирование и т. д. Но в этом трехтомнике подобные темы не рассматриваются (за исключением случаев, когда это необходимо сделать по ходу изложения). Численное компьютерное программирование — необычайно интересная и бурно развивающаяся область; на эту тему написано очень много хороших книг. Но с 60-х годов компьютеры все чаще и чаще применяются для решения проблем, в которых числа играют второстепенную роль. Теперь на первый план выходит способность компьютера принимать решения, а не просто выполнять арифметические операции. При решении нечисленных задач иногда требуется выполнять операции сложения и вычитания, но потребность в умножении и делении возникает довольно редко. Но, конечно, даже тот, кто в основном занимается численным компьютерным программированием, только выиграет от изучения нечисленных методов, так как они лежат и в основе числовых программ.

Результаты исследований в области нечисленного анализа разбросаны по многим техническим журналам. Моя цель состояла в том, чтобы извлечь из этого огромного объема информации только фундаментальные методы, которые можно применять в разнотипных ситуациях программирования. Я попытался обобщить выбранную информацию, чтобы получить то, что в большей или меньшей степени можно назвать "теорией", а также показать, как применять эту теорию при решении различных практических задач.

Конечно, "нечисленный анализ" — крайне неудачное название для данной области науки. Оно неудачно прежде всего потому, что содержит только отрицание

другого понятия; гораздо лучше было бы выбрать более содержательный термин, не имеющий приставки "не". Название "обработка информации" охватывает более широкую область, чем рассматриваемый здесь материал, а "методы программирования" — более узкую. Я считаю, что для темы, освещаемой в данных книгах, самым подходящим является название *анализ алгоритмов*, которое можно расшифровать как "теория свойств некоторых компьютерных алгоритмов".

Полный набор книг, озаглавленный как *Искусство программирования*, имеет следующую основную структуру.

Том 1. Основные алгоритмы

Глава 1. Основные понятия
Глава 2. Информационные структуры

Том 2. Получисленные алгоритмы

Глава 3. Случайные числа
Глава 4. Арифметика

Том 3. Сортировка и поиск

Глава 5. Сортировка
Глава 6. Поиск

Том 4. Комбинаторные алгоритмы

Глава 7. Комбинаторный поиск
Глава 8. Рекурсия

Том 5. Синтаксические алгоритмы

Глава 9. Лексикографический поиск
Глава 10. Синтаксический анализ

В томе 4 рассматривается очень большая тема, поэтому на самом деле он состоит из трех отдельных книг (томов 4A, 4B и 4C). Планируется также выпуск двух дополнительных томов по более специализированным темам: том 6, *Теория языков* (глава 11), и том 7, *Компиляторы* (глава 12).

Я приступил к этой работе в 1962 году с намерением написать единственную книгу, содержащую все перечисленные главы, но вскоре понял, что необходимо глубоко рассматривать выбранные темы, а не просто "скользить по поверхности". В результате получился текст такого объема, что материала каждой главы оказалось более чем достаточно для изучения в течение одного университетского семестра. И стало ясно, что необходимо разбить материал на несколько отдельных томов. Я знаю, что книга, содержащая только одну-две главы, выглядит довольно странно, но решил сохранить первоначальную нумерацию глав, чтобы упростить перекрестные ссылки. Планируется выпуск сокращенного варианта томов 1–5, который будет служить более общим справочником и/или учебником для студентов; в нем будет содержаться основная часть материала данных томов, а более специальная информация будет опущена. В сокращенном издании будет сохранена такая же нумерация глав, как и в полном.

Том 1 можно рассматривать как "пересечение" полного набора глав, в том смысле, что он содержит основные сведения, которые используются во всех остальных

книгах. С другой стороны, тома 2–5 можно читать независимо один от другого. Том 1 — это не только справочник, который необходимо использовать как пособие при чтении других томов; он может служить также университетским учебником либо пособием для самообразования по теме *структуры данных* (основное внимание следует уделить главе 2) или *дискретная математика* (основное внимание следует уделить разделам 1.1, 1.2, 1.3.3 и 2.3.4), или *программирование на языке машинных команд* (основное внимание следует уделить разделам 1.3 и 1.4).

Эти главы написаны с другой точки зрения, чем та, которая используется в самых современных книгах по программированию, т. е. я не пытался научить читателя пользоваться чужим программным обеспечением. Вместо этого я стремился научить читателя писать собственные программы более высокого качества.

Моя первоначальная цель заключалась в том, чтобы познакомить читателей с передовыми научными исследованиями в каждой из рассматриваемых областей знания. Но очень сложно постоянно быть в курсе дел отрасли, которая является экономически выгодной; бурный рост компьютерной науки сделал невозможным осуществление моей мечты. Образно говоря, я очутился на берегу безбрежного океана, содержащего десятки тысяч маленьких результатов, которые были получены десятками тысяч талантливых людей по всему миру. Поэтому мне пришлось поставить перед собой новую цель — сосредоточиться на "классических" методах, которые останутся актуальными в течение многих десятилетий, и описать их как можно лучше по мере моих возможностей. В частности, я попытался проследить историю каждой темы и заложить прочный фундамент ее дальнейшего развития. Я старался использовать точную терминологию, согласованную с той, которая применяется в современных публикациях, и попытался сообщить обо всех известных идеях последовательного программирования, выделяющихся простотой и изяществом формулировок.

Теперь несколько слов о математическом содержании данного многотомного издания. Материал излагается так, что он вполне доступен даже лицам со средним образованием; более сложные фрагменты они смогут просмотреть или просто опустить. В то же время те, кто имеют склонность к математике, смогут изучить интересные математические методы, связанные с дискретной математикой. Подобная двойственность представления информации была достигнута, с одной стороны, за счет присвоения рейтинга каждому упражнению (чтобы читатель смог отличать сложные с математической точки зрения упражнения от простых), а с другой стороны, благодаря такой организации разделов, при которой главные математические результаты сформулированы *перед* доказательствами. Доказательства предлагается либо провести самостоятельно в качестве упражнений (ответы на которые приводятся в отдельном разделе), либо найти в конце раздела.

Читатель, который, главным образом, интересуется программированием, а не математикой, может прекратить чтение раздела, как только математический материал станет слишком сложным для восприятия. С другой стороны, читатель-математик найдет для себя очень много интереснейших фактов. Многие математические публикации на тему программирования были ошибочными, поэтому одна из целей данной книги — предоставить читателю правильное математическое обоснование предмета изложения. И так как я считаю себя математиком, моя прямая

обязанность — правильно (насколько смогу) изложить материал с математической точки зрения.

Для чтения большей части математического материала вполне достаточно знания элементарной математики, так как почти вся остальная теория разрабатывается здесь же. Но иногда у меня возникает необходимость в более глубоких теоремах теории комплексного переменного, теории вероятностей, теории чисел и т. д. В подобных случаях я ссылаюсь на книги, содержащие подробное изложение данных тем.

Самое сложное решение, которое мне пришлось принять при подготовке этих книг, касалось способа представления различных методов. Преимущества блок-схем и пошаговых описаний алгоритмов хорошо известны; эти вопросы обсуждаются в статье "Computer-Drawn Flowcharts" в журнале ACM *Communications*, Vol. 6 (September 1963), pages 555–563. Для описания любого компьютерного алгоритма необходим также формальный и точный язык. Поэтому мне нужно было решить, какой язык использовать: алгебраический, такой как ALGOL или FORTRAN, либо машинно-ориентированный. Вероятно, многие сегодняшние компьютерные специалисты не согласятся с моим решением использовать машинно-ориентированный язык, но я убедился, что это был правильный выбор. На то существуют следующие причины.

a) На программиста большое влияние оказывает язык, на котором написаны программы. В настоящее время превалирует тенденция к выбору самых простых, а не самых оптимальных для компьютера конструкций языка. А программист, который знает машинно-ориентированный язык, стремится использовать более эффективные методы и таким образом создает более совершенные программы.

b) Все нужные нам программы, написанные на машинно-ориентированном языке, за редким исключением будут иметь небольшой размер. А это значит, что при наличии компьютера, обладающего минимальной вычислительной мощностью, проблем с использованием таких программ у нас не возникнет.

c) Языки высокого уровня не подходят для обсуждения важных деталей, имеющих отношение к низкому уровню, таких как связь сопрограмм, генерирование случайных чисел, арифметика высокой точности и многие другие проблемы, связанные с эффективным использованием памяти.

d) Тот, кто серьезно интересуется компьютерами, должен хорошо знать машинный язык, так как он лежит в основе работы компьютера.

e) Некоторое знание машинного языка необходимо в любом случае, чтобы разобраться в выходных данных программ, приведенных во многих примерах.

f) Новые алгебраические языки входят и выходят из моды приблизительно каждые пять лет, в то время как я пытаюсь говорить о "вечных истинах".

С другой стороны, я признаю, что писать программы на языках высокого уровня и отлаживать эти программы значительно проще. В сущности, начиная с 1970 года, я сам редко использовал машинный язык низкого уровня для собственных программ, так как современные компьютеры обладают большим объемом памяти и высоким быстродействием. Но для решения многих проблем, рассматриваемых в данной

книге, наибольшее значение имеет искусство программирования. Например, некоторые комбинаторные вычисления нужно повторять триллионы раз, и мы сэкономим приблизительно 11,6 дней работы за счет того, что сократим время вычислений во внутреннем цикле всего на одну микросекунду. Аналогично имеет смысл приложить дополнительные усилия для написания программы, которая будет использоваться много раз в течение каждого дня на множестве компьютеров, тем более что написать эту программу нужно только один раз.

А если принять решение использовать машинно-ориентированный язык, то какому языку следует отдать предпочтение? Я мог бы выбрать язык для конкретной машины X, но тогда те, кто используют другой компьютер, подумают, что данная книга написана только в расчете на обладателей компьютера X. Более того, машина X, вероятно, имеет много характерных особенностей, для которых совершенно неприменим материал данной книги, но все же его необходимо изложить. И наконец, через два года фирма — производитель машины X выпустит машину $X+1$ или $10X$, и компьютер X больше никого не будет интересовать.

Чтобы решить эту проблему, я попытался разработать "идеальный" компьютер с очень простыми правилами работы (изучить которые можно, скажем, всего за час) и очень похожий на реальные машины. Для студента нет никакой причины избегать изучения характеристик различных компьютеров; после изучения одного языка все остальные будут усваиваться гораздо легче. Кроме того, серьезный программист должен быть готов к тому, что в ходе работы ему придется сталкиваться с различными машинными языками. Поэтому остается только один недостаток использования вымышленной машины — сложность запуска написанных для нее программ. К счастью, на самом деле это не проблема, так как много добровольцев предложили свои услуги по написанию имитаторов гипотетической машины. Такие имитаторы идеальны для учебных целей, и работать с ними даже проще, чем с реальным компьютером.

Я старался ссылаться на самые лучшие старые статьи по каждой теме, а также упоминать новые работы. Ссылаясь на литературные источники, я использовал стандартные сокращения для названий периодических изданий, за исключением наиболее часто цитируемых журналов, для которых применялись следующие сокращения.

CACM — Communications of the Association for Computing Machinery

JACM — Journal of the Association for Computing Machinery

Comp. J. — The Computer Journal (British Computer Society)

Math. Comp. — Mathematics of Computation

AMM — American Mathematical Monthly

SICOMP — SIAM Journal on Computing

FOCS — IEEE Symposium on Foundations of Computer Science

SODA — ACM-SIAM Symposium on Discrete Algorithms

STOC — ACM Symposium on Theory of Computing

Crelle — Journal für die reine und angewandte Mathematik

Например, "*CACM* **6** (1963), 555–563" означает ссылку на журнал, упомянутый в одном из предыдущих абзацев этого предисловия. Сокращение "*CMath*" я использовал также для обозначения книги *Конкретная математика*, на которую есть ссылка во введении к разделу 1.2.

Бóльшая часть технического материала этих книг приходится на упражнения. Если идея нетривиального упражнения принадлежала не мне, то я старался упомянуть ее автора. Ссылки на литературу обычно даются в тексте раздела либо в ответе к упражнению. Но во многих случаях в основе упражнений лежат неопубликованные материалы, на которые нельзя дать ссылки.

В течение долгих лет работы над этими книгами мне помогали многие люди, которым я благодарен от всей души. Прежде всего, я хочу выразить благодарность моей жене Джилл) за ее бесконечное терпение, за подготовку некоторых иллюстраций и за постоянную помощь во всем. Я признателен также Роберту В. Флойду (Floyd Robert W.) за то, что в 60-х годах он посвятил столько времени работе над улучшением и углублением данного материала. Тысячи других людей также оказали мне неоценимую помощь. Чтобы просто перечислить их имена, понадобилась бы еще одна такая книга! Многие из них любезно разрешили мне использовать их старые неопубликованные работы. Мои исследования в Калифорнийском технологическом институте и Станфордском университете щедро финансировались Национальным научным фондом (National Science Foundation) и департаментом морских исследований (Office of Naval Research). Издательство Addison-Wesley всегда оказывало мне всемерную помощь и поддержку с того самого времени, когда в 1962 году я приступил к работе над проектом. Мне кажется, что для всех этих людей лучшая благодарность — данная публикация. Она показывает, что их вклад привел к появлению книг, в которых, я надеюсь, мне удалось написать то, чего они ожидали.

Предисловие к третьему изданию

Потратив десять лет на разработку систем компьютерного набора METAFONT и TEX, я теперь могу осуществить свою мечту — применить эти системы для набора книги *Искусство программирования*. Наконец-то мне удалось внести полный текст этой книги в персональный компьютер и таким образом получить ее электронную версию, что позволит в дальнейшем вносить любые изменения в технологию печати и отображения на экране. Такой способ работы дал мне возможность сделать буквально тысячи улучшений; я добился того, о чем так долго мечтал.

В этом новом издании я смог проверить каждое слово в тексте, стараясь сохранить юношеский задор моих первоначальных исследований и в то же время внести бóльшую зрелость суждений. Были добавлены десятки новых упражнений, а на десятки старых даны новые или улучшенные ответы.

Таким образом, работа над книгой *Искусство программирования* продолжается. Именно поэтому некоторые части данной книги начинаются пиктограммой "В процессе построения" (это своеобразное извинение за то, что приведены не самые новые данные). Мои папки переполнены важными материалами, которые я планирую включить в окончательное, славное четвертое издание тома 1; оно выйдет,

вероятно, через 15 лет. Но сначала я должен закончить тома 4 и 5. Я хочу, чтобы они были опубликованы сразу же, как только будут готовы к печати.

Бóльшая часть тяжелой работы по подготовке этого нового издания была выполнена Филлис Винклер (Phyllis Winkler) и Сильвио Леви (Silvio Levy), которые профессионально набирали и редактировали текст второго издания, а также Джеффри Олдхэмом (Jeffrey Oldham), который конвертировал почти все оригинальные иллюстрации в формат METAPOST. Я исправил все ошибки, которые бдительные читатели (Бэрри) обнаружили во втором издании (а также ошибки, которые, увы, не заметил никто), и постарался избежать появления новых ошибок в новом материале. Тем не менее я допускаю, что некоторые огрехи все же остались, и хотел бы исправить их как можно скорее. Поэтому за каждую опечатку*, а также ошибку, относящуюся к сути излагаемого материала или к приведенным историческим сведениям, я охотно заплачу $2,56 тому, кто первым ее найдет. На Web-странице, адрес которой приведен на обороте титульной страницы, содержится текущий список всех ошибок, о которых мне сообщили**.

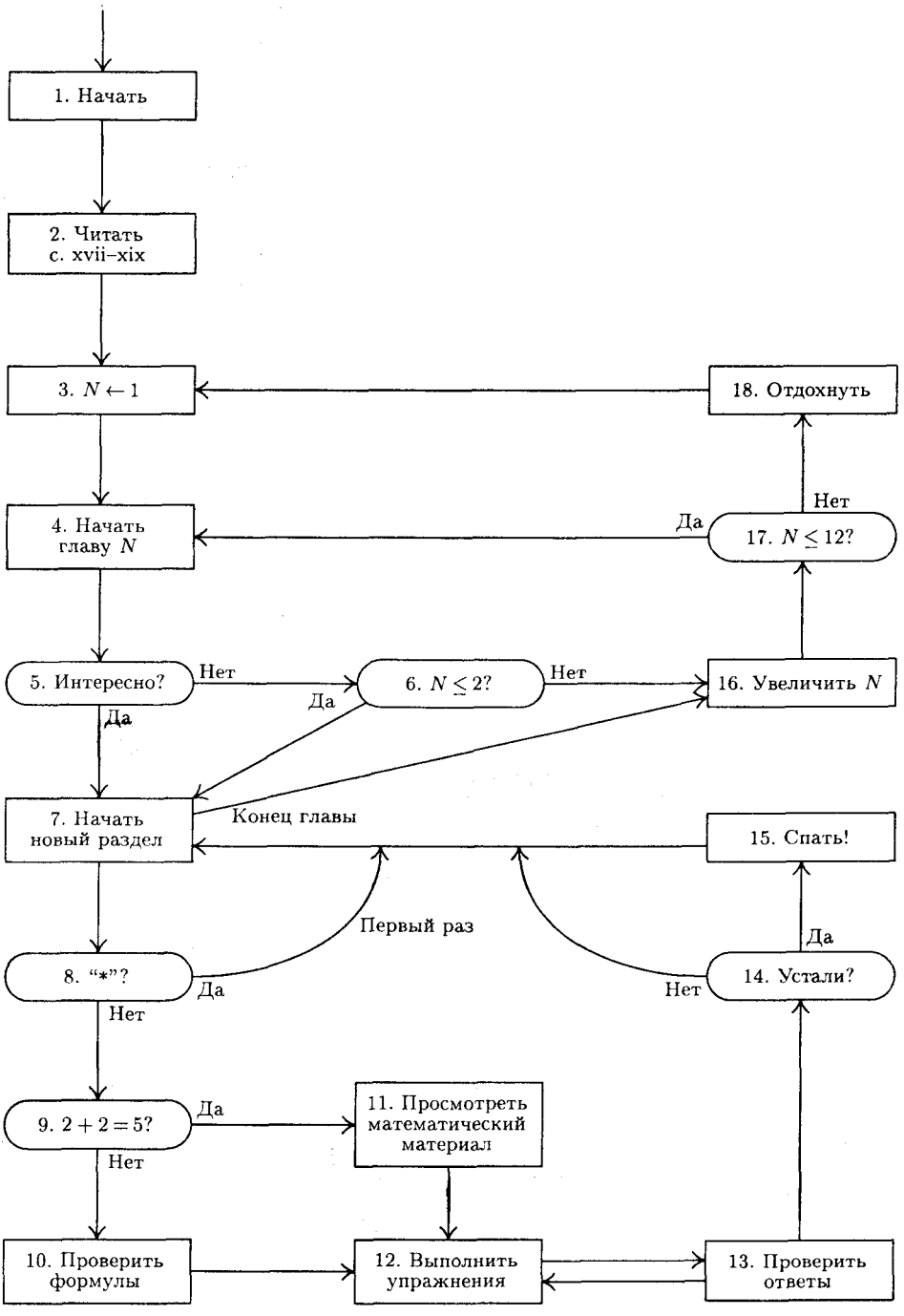
Станфорд, Калифорния
Апрель 1997

За последние двадцать лет мир изменился.

— Билл Гейтс (Bill Gates) (1995)

* Имеется в виду оригинал настоящего издания. — *Прим. ред.*

** Ошибки, известные на момент подготовки русского издания, были исправлены. — *Прим. ред.*



Блок-схема процедуры чтения книг этой серии.

Процедура чтения книг этой серии

1. Начните читать настоящую процедуру, если вы этого еще не сделали. *Продолжайте в точном соответствии с указанными шагами.* (Общая форма этой процедуры и сопровождающей ее блок-схемы будет использоваться на протяжении всей книги.)
2. Прочтите примечания к упражнениям (с. 23–25).
3. Установите N равным 1.
4. Начните чтение главы N. *Не* читайте эпиграфы, помещенные в ее начале.
5. Вам интересен предмет этой главы? Если да, перейдите к шагу 7; если нет, перейдите к шагу 6.
6. $N \leq 2$? Если нет, перейдите к шагу 16; если да, то все-таки просмотрите главу. (В главах 1 и 2 содержится важный вводный материал, а также обзор основных методов программирования. Эти главы следует хотя бы просмотреть, чтобы ознакомиться с условными обозначениями и компьютером MIX.)
7. Начните чтение следующего раздела главы; но, если вы уже дошли до конца главы, перейдите к шагу 16.
8. Отмечен ли номер раздела символом "*"? Если да, то при первом чтении этот раздел можно пропустить (в нем рассматривается специальный вопрос, который интересен, но не имеет первостепенного значения). Вернитесь к шагу 7.
9. Есть ли у вас наклонности к математике? Если математика для вас — китайская грамота, то перейдите к шагу 11; в противном случае перейдите к шагу 10.
10. Проверьте математические выкладки, выполненные в этом разделе (и сообщите автору о замеченных ошибках). Перейдите к шагу 12.
11. Если в текущем разделе содержится много математических выкладок, то лучше не читайте их. Тем не менее вам следует ознакомиться с основными результатами раздела; они, как правило, либо приводятся в самом начале раздела, либо выделены *курсивом* в самом конце сложной части текста.
12. Выполните рекомендуемые упражнения к разделу в соответствии с указаниями, приведенными в примечаниях к упражнениям (которые вы прочитали на шаге 2).
13. Поработав над упражнениями в свое удовольствие, сравните полученные ответы с теми, которые приведены в соответствующем разделе в конце книги (если для

этих задач даны ответы). Прочтите также ответы к упражнениям, над которыми у вас не было времени поработать. (*Замечание.* В большинстве случаев имеет смысл сначала прочесть ответ к упражнению n, а затем приступить к упражнению $n + 1$, поэтому шаги 12 и 13 обычно выполняются одновременно.)

14. Вы устали? Если нет, вернитесь к шагу 7.

15. Отправляйтесь спать. А когда проснетесь, вернитесь к шагу 7.

16. Увеличьте N на 1. Если $N = 3, 5, 7, 9, 11$ или 12, то возьмите следующий том этой серии книг.

17. Если N меньше или равно 12, то вернитесь к шагу 4.

18. Поздравляю! Теперь постарайтесь убедить друзей в том, что необходимо приобрести экземпляр тома 1 и начать его читать. Сами же возвращайтесь к шагу 3.

Горе тому, кто читает только одну книгу.

— ДЖОРДЖ ГЕРБЕРТ (GEORGE HERBERT), *Jacula Prudentum*, 1144 (1640)

Единственный недостаток всех литературных произведений в том, что они слишком длинны.

— ВОВЕНАРГ (VAUVENARGUES), *Réflexions*, 628 (1746)

Книги банальны. Гениальна только жизнь.

— ТОМАС КАРЛЕЙЛЬ (THOMAS CARLYLE), *Journal* (1839)

ПРИМЕЧАНИЯ К УПРАЖНЕНИЯМ

Упражнения, приведенные в этой серии книг, предназначены как для самостоятельной проработки, так и для семинарских занятий. Очень трудно и, наверное, просто невозможно выучить предмет, только читая теорию и не применяя ее для решения конкретных задач, которые заставляют задуматься о прочитанном. Более того, мы лучше всего заучиваем то, до чего дошли самостоятельно, своим умом. Поэтому упражнения занимают важное место в данном издании. Я приложил немало усилий, чтобы сделать их как можно более информативными, а также отобрать задачи, которые были бы не только поучительны, но и позволяли читателю получить удовольствие от их решения.

Во многих книгах простые упражнения даются вперемешку с исключительно сложными. Это не всегда удобно, так как читателю хочется знать заранее, сколько времени ему придется затратить на решение задач (иначе в лучшем случае он их только просмотрит). В качестве классического примера подобной ситуации можно привести книгу Ричарда Беллмана (Richard Bellman) *Динамическое программирование* (М.: Изд-во иностр. лит., 1960). Это очень важная, новаторская работа, но у нее есть один недостаток: в конце некоторых глав в разделе "Упражнения и научные проблемы" среди серьезных, еще нерешенных проблем, попадаются простейшие вопросы. Говорят, что кто-то однажды спросил д-ра Беллмана, как отличить упражнения от научных проблем, и он ответил: "Если вы можете решить задачу, значит, это упражнение; в противном случае это научная проблема".

Совершенно очевидно, что в книге, подобной этой, должны быть приведены и сложные научные проблемы, и простейшие упражнения. Поэтому, чтобы читатель не ломал голову, пытаясь отличить одно от другого, были введены рейтинги, которые определяют степень сложности каждого упражнения. Эти рейтинги имеют следующее значение.

Рейтинг *Объяснение*

00 Чрезвычайно простое упражнение, на которое можно ответить сразу же, если прочитанный материал понят. Упражнения подобного типа почти всегда можно решить "в уме".

10 Простая задача, которая заставляет задуматься над прочитанным, но не представляет особых трудностей. На ее решение вы затратите не больше минуты; в процессе решения могут понадобиться карандаш и бумага.

20 Средняя задача, которая позволяет проверить, понял ли читатель основные положения изложенного материала. Чтобы получить исчерпывающий ответ, может понадобиться примерно 15–20 минут.

30 Задача умеренной сложности. Для ее решения может понадобиться более двух часов (а если одновременно вы смотрите телевизор, то еще больше).

40 Достаточно сложная или трудоемкая задача, которую вполне можно включить в план семинарских занятий. Предполагается, что студент должен справиться с ней, затратив не слишком много времени, и решение будет нетривиальным.

50 Научная проблема, которая (насколько известно автору в момент написания книги) пока еще не получила удовлетворительного решения, хотя найти его пытались очень многие. Если вы нашли решение подобной проблемы, то опубликуйте его; более того, автор данной книги будет очень признателен, если ему сообщат решение как можно скорее (при условии, что оно правильно).

Интерполируя по этой "логарифмической" шкале, можно понять, что означает любой промежуточный рейтинг. Например, рейтинг *17* говорит о том, что упражнение немного проще, чем задача средней сложности. Если задача с рейтингом *50* будет впоследствии решена каким-либо читателем, то в следующих изданиях данной книги и в списке ошибок, опубликованных в Internet, она может иметь рейтинг *45* (адрес Web-страницы приводится на обороте титульной страницы).

Остаток от деления рейтинга на 5 показывает, какой объем рутинной работы потребуется для решения данной задачи. Таким образом, для решения упражнения с рейтингом *24* может потребоваться больше времени, чем для упражнения с рейтингом *25*, но для последнего необходим более творческий подход.

Автор очень старался правильно присвоить рейтинги упражнениям, но тому, кто составляет задачи, трудно предвидеть, насколько сложными они окажутся для кого-то другого. К тому же одному человеку некая задача может показаться простой, а другому — сложной. Таким образом, определение рейтингов — дело достаточно субъективное и относительное. Я надеюсь, что рейтинги помогут вам получить правильное представление о степени трудности задач, но их следует воспринимать в качестве ориентира, а не в качестве абсолюта.

Эта книга написана для читателей с различным уровнем математической подготовки и научного кругозора, поэтому некоторые упражнения рассчитаны исключительно на тех, кто серьезно интересуется математикой или занимается ею профессионально. Если рейтингу предшествует буква *M*, значит, математические понятия и обоснования используются в упражнении в большей степени, чем это необходимо тому, кто интересуется в основном программированием алгоритмов. Если же упражнение отмечено буквами *HM*, то для его решения необходимо знание высшей математики в большем объеме, чем дается в настоящей книге. Но пометка *HM* совсем необязательно означает, что упражнение трудное.

Перед некоторыми упражнениями стоит стрелка "▸", которая означает, что они особенно поучительны и их очень рекомендуется выполнить. Само собой разумеется, никто не ожидает, что читатель (или студент) будет решать *все* задачи, поэтому наиболее важные из них и были выделены. Но это ни в коем случае не означает, что другие упражнения выполнять не стоит! Каждый читатель должен хотя бы попытаться решить все задачи, рейтинг которых меньше или равен *10*. Стрелки помогут выбрать задачи с более высокими рейтингами, которые следует решать в первую очередь.

К большинству упражнений приведены ответы, помещенные в отдельном разделе в конце книги. Пожалуйста, пользуйтесь ими разумно: ответ смотрите только

после того, как приложите все усилия, чтобы решить задачу самостоятельно, либо если у вас совершенно нет времени на ее решение. Ответ будет поучителен и полезен для вас только в том случае, если вы ознакомитесь с ним *после* того, как найдете свое решение или изрядно потрудитесь над задачей. Ответы к задачам излагаются очень кратко и схематично, так как предполагается, что читатель честно пытался решить задачу собственными силами. Иногда в приведенном решении дается меньше информации, чем спрашивалось, но чаще бывает наоборот. Вполне возможно, что полученный вами ответ окажется лучше того, который помещен в книге, или вы найдете ошибку в ответе. В таком случае автор был бы очень признателен, если бы вы как можно скорее подробно сообщили ему об этом; тогда в последующих изданиях книги будет опубликовано более удачное решение, а также имя его автора.

Решая задачи, вы, как правило, можете пользоваться ответами к предыдущим упражнениям, за исключением случаев, когда это будет оговорено особо. Рейтинги упражнениям присваивались в расчете именно на это, и вполне возможно, что рейтинг упражнения $n + 1$ ниже рейтинга упражнения n, даже если результат упражнения n является его частным случаем.

Условные обозначения		*00*	Простейшее (ответ дать немедленно)
		10	Простое (на одну минуту)
▶	Рекомендуется	*20*	Средней трудности (на четверть часа)
M	С математическим уклоном	*30*	Повышенной трудности
HM	Требует знания высшей математики	*40*	Высокой трудности
		50	Научная проблема

УПРАЖНЕНИЯ

▶ **1.** [*00*] Что означает рейтинг *M20*?

2. [*10*] Какое значение для читателя имеют упражнения, которые приводятся в учебниках?

3. [*14*] Докажите, что $13^3 = 2197$. Обобщите ответ. (Скучных задач, подобных этой, автор старался избегать.)

4. [*HM45*] Докажите, что если n — целое число, $n > 2$, то уравнение $x^n + y^n = z^n$ неразрешимо в целых положительных числах x, y, z.

Мы обсудили этот вопрос со всех сторон.
Перед нами факты, изложенные систематично и по порядку.

— ЭРКЮЛЬ ПУАРО (HERCULE POIROT),
Убийство в восточном экспрессе (1934)

ОСНОВНЫЕ ПОНЯТИЯ

Многие не сведущие
в математике люди
думают, что поскольку назначение
аналитической машины Бэббиджа (Babbage) —
выдавать результаты в численном виде,
то природа происходящих в ней процессов
должна быть арифметической и численной,
а не алгебраической и аналитической.
Но они ошибаются. Машина может упорядочивать
и комбинировать числовые значения
так же, как и буквы или любые другие символы
общего характера. В сущности, при выполнении
соответствующих условий она могла бы
выдавать результаты и в алгебраическом виде.

— АВГУСТА АДА (AUGUSTA ADA), графиня Лавлейс* (Lovelace) (1844)

Прошу вас, ради всего святого, сначала научитесь простому
и только потом переходите к сложному.

— ЭПИКТЕТ (EPICTETUS), *Беседы* IV.i

1.1. АЛГОРИТМЫ

Понятие *алгоритм* является основным для всей области компьютерного программирования, поэтому начать мы должны с тщательного анализа этого термина. Слово "алгоритм" (algorithm) (иногда используется устаревшее слово "алгорифм". — *Прим. перев.*) уже само по себе представляет большой интерес. На первый взгляд может показаться, будто кто-то собирался написать слово "логарифм" (logarithm), но случайно переставил первые четыре буквы. Этого слова еще не было в издании словаря *Webster's New World Dictionary*, вышедшем в 1957 году. Мы находим там только устаревшую форму "algorism" — старинное слово, которое означает "выполнение арифметических действий с помощью арабских цифр". В средние века абакисты считали на абаках (счетных досках), а алгоритмики использовали "algorism". В эпоху Возрождения происхождение этого слова оказалось забытым. Одни лингвисты того времени пытались объяснить его значение путем сочетания

* Дочь великого английского поэта Дж. Г. Байрона, которую считают основоположницей программирования. Большинство идей и принципов программирования для аналитической машины Бэббиджа было рассмотрено в ее книге "Комментарии". *Прим. перев.*

слов *algiros* [больной] и *arithmas* [число], другие не соглашались с таким толкованием и утверждали, что это слово происходит от "King Algor of Castile". Наконец историки математики обнаружили истинное происхождение слова "algorism": оно берет начало от имени автора знаменитого персидского учебника по математике, Abū 'Abd Allāh Muḥammad ibn Mūsā al-Khwārizmī (Абу Абд Аллах Мухаммед ибн Муса аль-Хорезми) (ок. 825 г.), означающего буквально "Отец Абдуллы, Мухаммед, сын Мусы, уроженец Хорезма"*. Аральское море в Центральной Азии когда-то называлось озером Хорезм, и район Хорезма (Khwārizm) расположен в бассейне реки Амударьи южнее этого моря. Аль-Хорезми написал знаменитую книгу *Kitāb al-jabr wa'l-muqābala* (Китаб аль-джебр валь-мукабала — "Книга о восстановлении и противопоставлении"). От названия этой книги, которая была посвящена решению линейных и квадратных уравнений, произошло еще одно слово — "алгебра". [О жизни и научной деятельности аль-Хорезми речь идет в работе H. Zemanek, *Lecture Notes in Computer Science* **122** (1981), 1–81.]

Постепенно форма и значение слова *algorism* исказились; как объясняется в словаре *Oxford English Dictionary*, это слово "претерпело множество псевдоэтимологических искажений, включая последний вариант *algorithm*, где произошла путаница" с корнем слова греческого происхождения *arithmetic*. Этот переход от "algorism" к "algorithm" кажется вполне закономерным ввиду того, что происхождение рассматриваемого слова было полностью забыто. В старинном немецком математическом словаре *Vollständiges mathematisches Lexicon* (Leipzig, 1747) дается следующее определение слова *algorithmus*: "Этот термин включает в себя понятие о четырех типах арифметических операций, а именно: о сложении, умножении, вычитании и делении". Латинское выражение *algorithmus infinitesimalis* в то время использовалось для определения "способов выполнения действий с бесконечно малыми величинами, открытых Лейбницем (Leibniz)".

К 1950 году слово "алгоритм" чаще всего ассоциировалось с алгоритмом Евклида, который представляет собой процесс нахождения наибольшего общего делителя двух чисел. Этот алгоритм приведен в книге Евклида (Euclid) *Начала* (книга 7, предложения 1 и 2). Думаю, имеет смысл привести здесь описание этого алгоритма.

Алгоритм E (*Алгоритм Евклида*). Даны два целых положительных числа m и n. Требуется найти их *наибольший общий делитель*, т. е. наибольшее целое положительное число, которое нацело делит оба числа m и n.

E1. [Нахождение остатка.] Разделим m на n, и пусть остаток от деления будет равен r (где $0 \le r < n$).

E2. [Сравнение с нулем.] Если $r = 0$, то выполнение алгоритма прекращается; n — искомое значение.

E3. [Замещение.] Присвоить $m \leftarrow n$, $n \leftarrow r$ и вернуться к шагу E1. ▮

Разумеется, у Евклида этот алгоритм сформулирован не совсем так. Приведенная выше формулировка иллюстрирует стиль, в котором алгоритмы будут представлены на протяжении всей этой книги.

Каждому рассматриваемому алгоритму присваивается идентифицирующая буква (в предыдущем примере использовалась буква E), а шагам алгоритма — эта

* В русскоязычных источниках это историческое лицо обычно упоминается под именем "аль-Хорезми". — *Прим. перев.*

же буква в сочетании с числом (E1, E2, E3). Главы книги подразделяются на пронумерованные разделы; внутри раздела алгоритмы обозначаются только буквами. Но когда на эти алгоритмы делаются ссылки из других разделов, то к букве присоединяется номер соответствующего раздела. Например, сейчас мы находимся в разделе 1.1; внутри этого раздела алгоритм Евклида называется "Алгоритм E", но ссылаться на него в последующих разделах мы будем как на алгоритм 1.1E.

Каждый шаг любого алгоритма, например E1 в вышеприведенном алгоритме, начинается заключенной в квадратные скобки фразой, которая как можно более кратко выражает содержание данного шага. Обычно эта фраза отражается также в сопровождающей алгоритм *блок-схеме*, такой как на рис. 1, чтобы читатель мог легко представить себе описанный алгоритм.

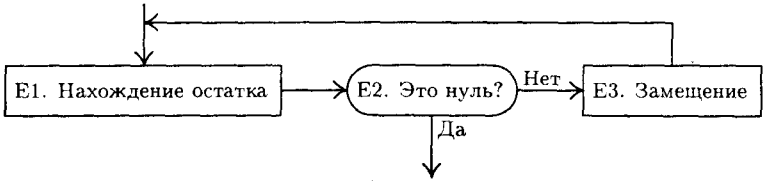


Рис. 1. Блок-схема алгоритма E.

За краткой фразой следует формулировка (выраженная с помощью слов и символов) *действия*, которое нужно выполнить, или решения, которое нужно принять. Могут присутствовать также заключенные в круглые скобки *комментарии* (например, второе предложение шага E1). Комментарии играют роль пояснений к шагу; с их помощью часто указываются некоторые постоянные характеристики переменных или текущие цели данного этапа. В комментариях не определяются действия, которые являются составной частью алгоритма; они служат только для удобства читателя, чтобы по возможности помочь ему разобраться в алгоритме.

Стрелка "$\leftarrow$", используемая на шаге E3, обозначает важнейшую операцию *замещения*, которую иногда называют *присвоением* или *подстановкой*: запись $m \leftarrow n$ указывает, что значение переменной m замещается текущим значением переменной n. В начале работы алгоритма E m и n — это заданные первоначальные значения, но по окончании его работы эти переменные будут иметь, вообще говоря, совершенно другие значения. Стрелка используется для того, чтобы отличать операцию замещения от отношения равенства. Мы не будем говорить: "Установим $m = n$", а, вероятно, спросим: "Действительно ли $m = n$?". Знак "$=$" обозначает условие, которое можно проверить, а знак "$\leftarrow$" — действие, которое можно выполнить. Операция *увеличение n на единицу* обозначается через $n \leftarrow n + 1$ (и читается так: "n замещается значением $n + 1$" или "n принимает значение $n + 1$"). Вообще говоря, запись "переменная $\leftarrow$ формула" означает, что формула будет вычислена на основании текущих значений всех входящих в нее переменных, а полученный результат будет присвоен переменной, стоящей слева от стрелки (таким образом, вычисленный по формуле результат заменит собой предыдущее значение переменной слева). Лица, не имеющие достаточного опыта программирования, иногда говорят, что "n переходит в $n + 1$" и для обозначения операции увеличения n на единицу

используют запись $n \to n+1$. Такая система обозначений и формулировок противоречит стандартным соглашениям и может привести только к путанице, поэтому ее следует избегать.

Обратите внимание, что на шаге E3 очень важен порядок действий. Действительно, записи "присвоить $m \leftarrow n$, $n \leftarrow r$" и "присвоить $n \leftarrow r$, $m \leftarrow n$" — это совсем не одно и то же. Из второй записи следует, что предыдущее значение n будет потеряно до того, как его смогут присвоить m. На самом деле эквивалентом второй записи будет "присвоить $n \leftarrow r$, $m \leftarrow r$". Когда нескольким переменным присваивается одно и то же значение, в одном выражении можно использовать несколько стрелок. Так, например, операцию "$n \leftarrow r$, $m \leftarrow r$" можно записать как "$n \leftarrow m \leftarrow r$". Операцию взаимного обмена значениями двух переменных можно записать так: "Обмен $m \leftrightarrow n$". Ее можно записать и с помощью новой переменной t следующим образом: "Присвоить $t \leftarrow m$, $m \leftarrow n$, $n \leftarrow t$".

Выполнение алгоритма начинается с шага, имеющего наименьший номер (обычно это шаг 1). Затем последовательно выполняются следующие шаги, если нет каких-либо указаний, нарушающих естественный порядок выполнения. На шаге E3 указание "Вернуться к шагу E1" явным образом определяет порядок вычислений. На шаге E2 действию предшествует условие "Если $r = 0$" и если $r \neq 0$, то оставшаяся часть предложения не применяется и нет указаний на выполнение в этом случае каких-либо действий. Конечно, мы могли бы добавить дополнительное предложение "Если $r \neq 0$, то перейти к шагу E3", но это совершенно излишне.

Жирная вертикальная черточка "∎", помещенная в конце шага E3, обозначает окончание алгоритма и продолжение текста.

Итак, мы обсудили практически все соглашения об обозначениях, которые используются в алгоритмах, приведенных в книге. Осталось выяснить только, как обозначать величины с индексами (или "подстрочными" индексами), которые являются элементами упорядоченного массива. Предположим, у нас есть n величин: $v_1, v_2, \ldots, v_n$. Для обозначения j-го элемента вместо записи v_j часто используется запись $v[j]$. Аналогично массив иногда предпочитают обозначать как $a[i, j]$, вместо того чтобы использовать два подстрочных индекса, как в записи a_{ij}. Иногда для обозначения переменных используются имена, состоящие из нескольких букв, обычно прописных. Например, `TEMP` может быть именем переменной, использующейся для временного хранения вычисленного значения, а `PRIME[K]` может обозначать K-е простое число, и т. д.

До сих пор мы говорили о *форме записи* алгоритмов, а теперь давайте попробуем *выполнить* один из них. Хочу сразу заметить, что читателю *не* следует рассчитывать на то, что алгоритмы можно читать, как роман. Такое чтение приведет к тому, что вам будет трудно понять, что же на самом деле происходит при выполнении алгоритма. Чтобы проверить алгоритм, в нем нужно разобраться, и лучший способ понять, как он работает, — испытать его. Поэтому я предлагаю вам взять карандаш и бумагу и прорабатывать от начала до конца каждый алгоритм сразу же, как только он встретится в тексте. Обычно к примеру алгоритма прилагается схема, в противном случае читатель легко сможет представить ее. Это самый простой и доступный способ разобраться в алгоритме, в то время как все остальные подходы оказываются неэффективными.

Итак, давайте в качестве примера разберем алгоритм Е. Предположим, что $m = 119$ и $n = 544$; начнем с шага Е1. (Сейчас можете просто следить за изложением, так как мы разберем алгоритм и подробно все выпишем.) Деление m на n в этом случае выполняется просто, даже очень просто, так как частное равно нулю, а остаток — 119. Таким образом, $r \leftarrow 119$. Переходим к шагу Е2. Поскольку $r \neq 0$, на этом шаге никакие действия не выполняются. На шаге Е3 присваиваем $m \leftarrow 544$, $n \leftarrow 119$. Очевидно, что если первоначально $m < n$, то частное на шаге Е1 всегда оказывается равным нулю и в ходе выполнения алгоритма всегда происходит взаимный обмен значений переменных m и n, хотя и таким громоздким способом. Поэтому можно добавить дополнительный шаг.

Е0. [Гарантировать, что $m \geq n$.] Если $m < n$, то выполнить взаимный обмен $m \leftrightarrow n$.

В результате алгоритм изменится незначительно (разве что увеличится на один шаг), но зато время его выполнения сократится примерно в половине случаев.

Вернемся к шагу Е1. Находим, что $\frac{544}{119} = 4\frac{68}{119}$, поэтому $r \leftarrow 68$. В результате на шаге Е2 снова не выполняются никакие действия, а на шаге Е3 присваиваем $m \leftarrow 119$, $n \leftarrow 68$. В следующих циклах сначала получаем $r \leftarrow 51$ и $m \leftarrow 68$, $n \leftarrow 51$, а затем — $r \leftarrow 17$ и $m \leftarrow 51$, $n \leftarrow 17$. Наконец, в результате деления 51 на 17 получаем $r \leftarrow 0$. Таким образом, на шаге Е2 выполнение алгоритма прекращается. Наибольший общий делитель 119 и 544 равен 17.

Вот что такое алгоритм. Современное значение слова "алгоритм" во многом аналогично таким понятиям, как *рецепт, процесс, метод, способ, процедура, программа*, но все-таки слово "algorithm" имеет дополнительный смысловой оттенок. Алгоритм — это не просто набор конечного числа правил, задающих последовательность выполнения операций для решения задачи определенного типа. Помимо этого, он имеет пять важных особенностей.

1) *Конечность.* Алгоритм всегда должен заканчиваться после выполнения конечного числа шагов. Алгоритм Е удовлетворяет этому условию, потому что после шага Е1 значение r *меньше*, чем n. Поэтому если $r \neq 0$, то в следующем цикле на шаге Е1 значение n *уменьшается*. Убывающая последовательность положительных целых чисел имеет конечное число членов, поэтому шаг Е1 может выполняться только конечное число раз для любого первоначально заданного значения n. Но имейте в виду, что количество шагов может быть сколь угодно большим; выбор слишком больших значений m и n приведет к тому, что шаг Е1 будет выполняться более миллиона раз.

Процедура, обладающая всеми характеристиками алгоритма, за исключением, возможно, конечности, называется *методом вычислений*. Евклид (Euclid) предложил не только алгоритм нахождения наибольшего общего делителя, но и аналогичное ему геометрическое построение "наибольшей общей меры" длин двух отрезков прямой; это уже метод вычислений, выполнение которого не заканчивается, если заданные длины оказываются несоизмеримыми.

2) *Определенность.* Каждый шаг алгоритма должен быть точно определен. Действия, которые нужно выполнить, должны быть строго и недвусмысленно определены для каждого возможного случая. Я надеюсь, что алгоритмы, приведенные в данной книге, удовлетворяют этому критерию. Но дело в том, что они описываются обычным языком, и поэтому существует возможность неточного понимания

читателем мысли автора. Чтобы преодолеть это затруднение, для описания алгоритмов были разработаны формально определенные *языки программирования*, или *машинные языки*, в которых каждый оператор имеет строго определенное значение. Многие алгоритмы в этой книге даются как на обычном языке, так и на языке программирования. Метод вычислений, выраженный на языке программирования, называется *программой*.

Рассмотрим в качестве примера алгоритм Е. Применительно к шагу Е1 критерий определенности означает, что читатель обязан точно понимать, что значит разделить m на n и что такое остаток. Но в действительности нет никакого общепринятого соглашения по поводу того, что это означает, если m и n не являются целыми положительными числами. Каким будет остаток от деления -8 на $-\pi$? Что понимать под остатком от деления $59/13$ на нуль? Поэтому в данном случае критерий определенности означает следующее: мы должны быть уверены, что в каждом случае выполнения шага Е1 значениями m и n всегда будут целые положительные числа. Если сначала по предположению это верно, то после шага Е1 r — это целое неотрицательное число; при условии перехода к шагу Е3 оно является также ненулевым. Таким образом, поставленное требование выполнено и m и n — это действительно целые положительные числа.

3) *Ввод.* Алгоритм имеет некоторое (возможно, равное нулю) число *входных данных*, т. е. величин, которые задаются до начала его работы или определяются динамически во время его работы. Эти входные данные берутся из определенного набора объектов. Например, в алгоритме Е есть два входных значения, а именно — m и n, которые принадлежат множеству *целых положительных* чисел.

4) *Вывод.* У алгоритма есть одно или несколько *выходных данных*, т. е. величин, имеющих вполне определенную связь с входными данными. У алгоритма Е имеется только одно выходное значение, а именно — n, получаемое на шаге Е2. Это наибольший общий делитель двух входных значений.

(Можно легко *доказать*, что это число действительно является наибольшим общим делителем. После шага Е1 имеем

$$m = qn + r,$$

где q — некоторое целое число. Если $r = 0$, то m кратно n и, очевидно, в этом случае n — наибольший общий делитель m и n. Если $r \neq 0$, то любой делитель обоих чисел m и n должен быть также делителем $m - qn = r$ и любой делитель n и r — также делителем $qn + r = m$. Таким образом, множество делителей чисел $\{m, n\}$ совпадает с множеством делителей чисел $\{n, r\}$. Следовательно, пары чисел $\{m, n\}$ и $\{n, r\}$ имеют один и тот же *наибольший* общий делитель. Таким образом, шаг Е3 не изменяет ответа исходной задачи.)

5) *Эффективность.* Алгоритм обычно считается *эффективным*, если все его операторы достаточно просты для того, чтобы их можно было точно выполнить в течение конечного промежутка времени с помощью карандаша и бумаги. В алгоритме Е используются только следующие операции: деление одного целого положительного числа на другое, сравнение с нулем и присвоение одной переменной значения другой. Эти операции являются эффективными, так как целые числа можно представить на бумаге с помощью конечного числа знаков и так как существует по меньшей

мере один способ ("алгоритм деления") деления одного целого числа на другое. Но те же самые операции были бы *неэффективными*, если бы они выполнялись над действительными числами, представляющими собой бесконечные десятичные дроби, либо над величинами, выражающими длины физических отрезков прямой, которые нельзя измерить абсолютно точно. Приведем еще один пример неэффективного шага: "Если 4 — это наибольшее целое n, при котором существует решение уравнения $w^n + x^n + y^n = z^n$ для целых положительных чисел w, x, y и z, то перейти к шагу Е4". Подобная операция не может считаться эффективной до тех пор, пока кто-либо не разработает алгоритм, позволяющий определить, действительно ли 4 является наибольшим целым числом с требуемым свойством.

Давайте попробуем сравнить понятие "алгоритм" с рецептом из кулинарной книги. Предполагается, что рецепт обладает свойством конечности (хотя и говорят, что "кто над чайником стоит, у того он не кипит"), имеет входные данные (такие, например, как яйца, мука и т. д.) и выходные данные (обед "на скорую руку" и т. п.), но хорошо известно, что ему не хватает определенности. Инструкции из кулинарных рецептов очень часто бывают неопределенными, например: "Добавьте щепотку соли". "Щепотка" определяется как количество, "меньшее $1/8$ чайной ложки", и что такое соль, вероятно, тоже известно всем. Но куда именно нужно добавить соль — сверху? сбоку? Инструкции "Слегка потрясите, пока смесь не станет рассыпчатой" и "Подогрейте коньяк в маленькой кастрюльке" будут вполне понятны опытному повару, но они не годятся для алгоритма. Алгоритм должен быть определен настолько четко, чтобы его указаниям мог следовать даже компьютер. Тем не менее программист может многому научиться, прочитав хорошую поваренную книгу. (Честно говоря, автор едва устоял перед искушением назвать настоящий том "Поваренная книга программиста". Но, может, когда-нибудь он попытается написать книгу под названием "Алгоритмы для кухни".)

Следует отметить, что для практических целей ограничение, состоящее в конечности алгоритма, в сущности, является недостаточно жестким. Используемый на практике алгоритм должен иметь не просто конечное, а *достаточно ограниченное*, разумное число шагов. Например, существует алгоритм определения того, может ли игра в шахматы всегда быть выиграна белыми при условии, что не было сделано ни одной ошибки (см. упр. 2.2.3–28). Этот алгоритм позволил бы решить проблему, представляющую огромный интерес для тысяч людей, но можно биться об заклад, что окончательный ответ на данный вопрос мы не узнаем никогда. Все дело в том, что для выполнения указанного алгоритма требуется невероятно большой промежуток времени, хотя сам алгоритм и является конечным. В главе 8 будут обсуждаться конечные числа, которые велики настолько, что, в сущности, находятся за пределами нашего понимания*.

На практике нам нужны не просто алгоритмы, а *хорошие* алгоритмы в широком смысле этого слова. Одним из критериев качества алгоритма является время, необходимое для его выполнения; данную характеристику можно оценить по тому, сколько раз выполняется каждый шаг. Другими критериями являются адаптируемость алгоритма к различным компьютерам, его простота, изящество и т. д.

Часто решить одну и ту же проблему можно с помощью нескольких алгоритмов и нужно выбрать наилучший из них. Таким образом, мы попадаем в чрезвычайно

* Глава 8 не входит в данное трехтомное издание. — *Прим. ред.*

интересную и крайне важную область *анализа алгоритмов*. Предмет этой области состоит в том, чтобы для заданного алгоритма определить рабочие характеристики.

В качестве примера давайте исследуем с этой точки зрения алгоритм Евклида. Предположим, нам нужно решить следующую задачу: "Пусть задано значение n, а m может быть любым целым положительным числом. Тогда чему равно *среднее* число T_n выполнений шага E1 алгоритма E?". Прежде всего необходимо убедиться в том, что задача имеет смысл, поскольку нам предстоит найти среднее при бесконечно большом количестве значений m. Но совершенно очевидно, что после первого выполнения шага E1 значение будет иметь только остаток от деления m на n. Поэтому все, что мы должны сделать для нахождения значения T_n, — это испытать алгоритм для $m = 1$, $m = 2$, ..., $m = n$, подсчитать суммарное число выполнений шага E1 и разделить его на n.

А теперь рассмотрим еще один важный вопрос, касающийся *поведения* T_n как функции от n: можно ли ее аппроксимировать, например, функцией $\frac{1}{3}n$ или $\sqrt{n}$? На самом деле это чрезвычайно сложная и интересная математическая проблема, которая еще не решена окончательно; более подробно она будет рассмотрена в разделе 4.5.3. Можно доказать, что при больших значениях n T_n ведет себя, как функция $\left(12(\ln 2)/\pi^2\right)\ln n$, т. е. она пропорциональна *натуральному логарифму* n. Заметим, что коэффициент пропорциональности нельзя просто взять и угадать; чтобы определить его, нужно затратить определенные усилия. Более подробно об алгоритме Евклида, а также о других способах вычисления наибольшего общего делителя будет говориться в разделе 4.5.2.

Для обозначения области подобных исследований автор использует термин *анализ алгоритмов*. Основная идея заключается в том, чтобы взять конкретный алгоритм и определить его количественные характеристики. Время от времени мы будем также выяснять, является ли алгоритм оптимальным в некотором смысле. *Теория алгоритмов* — это совершенно другая область, в которой, в первую очередь, рассматриваются вопросы существования или не существования эффективных алгоритмов вычисления определенных величин.

До сих пор наше обсуждение алгоритмов носило достаточно общий характер, и, вероятно, "математически настроенный" читатель утвердился в мысли, что все предыдущие комментарии представляют собой очень шаткий фундамент для построения какой-либо теории алгоритмов. Поэтому давайте подведем итог данного раздела, кратко описав метод, с помощью которого понятие алгоритма можно строго обосновать в терминах математической теории множеств. Формально определим *метод вычислений* как четверку (Q, I, Ω, f), где Q — это множество, содержащее подмножества I и Ω, а f — функция, переводящая множество Q в себя. Кроме того, f оставляет неподвижными точки множества Ω, т. е. $f(q)$ равно q для всех элементов q из множества Ω. Эти четыре элемента, Q, I, Ω, f, представляют соответственно состояния вычисления, ввод, вывод и правило вычислений. Каждое входное значение x из множества I определяет *вычисляемую последовательность* $x_0, x_1, x_2, \ldots$ следующим образом:

$$x_0 = x \qquad \text{и} \qquad x_{k+1} = f(x_k) \quad \text{для} \quad k \geq 0. \tag{1}$$

Говорят, что вычисляемая последовательность *заканчивается через k шагов*, если k — это наименьшее целое число, для которого x_k принадлежит Ω, и что она дает

выходное значение x_k для заданного x. (Заметим, что если x_k принадлежит Ω, то и x_{k+1} принадлежит Ω, так как в этом случае $x_{k+1} = x_k$.) Некоторые вычисляемые последовательности могут никогда не заканчиваться, но *алгоритм* — это метод вычислений, который заканчивается через конечное число шагов для всех x из I.

Например, алгоритм E в этих терминах можно формализовать следующим образом. Пусть элементами множества Q будут все величины (n), все упорядоченные пары (m,n) и все упорядоченные четверки $(m,n,r,1)$, $(m,n,r,2)$ и $(m,n,p,3)$, где m, n и p — это целые положительные числа, а r — неотрицательное целое число. Пусть I — это подмножество всех пар (m,n), а Ω — подмножество всех величин (n). Определим функцию f следующим образом:

$$\begin{aligned} f\big((m,n)\big) &= (m,n,0,1); \qquad f\big((n)\big) = (n);\\ f\big((m,n,r,1)\big) &= (m,\ n,\ \text{остаток от деления } m \text{ на } n,\ 2);\\ f\big((m,n,r,2)\big) &= (n), \quad \text{если} \quad r=0, \qquad (m,n,r,3) \quad \text{в противном случае;}\\ f\big((m,n,p,3)\big) &= (n,p,p,1). \end{aligned} \tag{2}$$

Соответствие между данной записью и алгоритмом E очевидно.

В этой формулировке понятия "алгоритм" не содержится ограничение, касающееся эффективности, о котором упоминалось ранее. Например, Q может быть множеством бесконечных последовательностей, которые нельзя вычислить с помощью карандаша и бумаги, а f может включать операции, которые простой смертный сможет выполнить не всегда. Если мы хотим ограничить понятие "алгоритм" таким образом, чтобы в нем могли содержаться только элементарные операции, то введем ограничения на элементы Q, I, Ω и f, например, следующим образом. Пусть A — это ограниченное множество букв, а A^* — множество всех строк, определенных на множестве A (т. е. множество всех упорядоченных последовательностей $x_1x_2\ldots x_n$, где $n \geq 0$ и x_j принадлежит A для $1 \leq j \leq n$). Идея заключается в следующем: закодировать состояния вычисления таким образом, чтобы они были представлены строками из множества A^*. Теперь пусть N — целое неотрицательное число, а Q — множество всех пар (σ, j), где σ принадлежит A^*, а j — целое число, $0 \leq j \leq N$. Пусть I — подмножество пар из Q, для которых $j = 0$, а Ω — подмножество пар из Q, для которых $j = N$. Если θ и σ — строки из A^*, то мы будем говорить, что θ входит в σ, если σ имеет вид $\alpha\theta\omega$, где α и ω — некоторые строки. И в завершение определим функцию f с помощью строк θ_j, ϕ_j и целых чисел a_j, b_j, $0 \leq j < N$ следующим образом:

$$\begin{aligned} &f(\sigma,j) = (\sigma, a_j), && \text{если } \theta_j \text{ не входит в } \sigma;\\ &f(\sigma,j) = (\alpha\phi_j\omega, b_j), && \text{если } \alpha \text{ является самой короткой строкой,}\\ &&& \text{для которой } \sigma = \alpha\theta_j\omega;\\ &f(\sigma,N) = (\sigma,N). \end{aligned} \tag{3}$$

Метод вычислений, удовлетворяющий этому определению, безусловно, является эффективным. Кроме того, опыт показывает, что в таком виде можно представить любую задачу, которая решается с помощью карандаша и бумаги. Существует также много других по сути эквивалентных способов формулировки понятия эффективного метода вычислений (например, с помощью машины Тьюринга). Приведенная выше формулировка практически совпадает с той, которую дал А. А. Марков

(Markov) в своей книге *Теория алгорифмов* [*Труды АН СССР, Ин-т математики. — 1954. — 42. — 376 с.*], а впоследствии исправил и расширил Н. М. Нагорный (Nagorny) (М.: Наука, 1984).

УПРАЖНЕНИЯ

1. [*10*] В тексте показано, как взаимно заменить значения переменных m и n с помощью символа замены, а именно — полагая $t \leftarrow m$, $m \leftarrow n$, $n \leftarrow t$. Покажите, как в результате ряда замен можно преобразовать *четверку* переменных (a, b, c, d) в (b, c, d, a). Другими словами, новое значение переменной a должно стать равным первоначальному значению b и т. д. Постарайтесь выполнить преобразование с помощью минимального числа замен.

2. [*15*] Докажите, что в начале выполнения шага E1 m всегда больше n, за исключением, возможно, только первого случая выполнения этого шага.

3. [*20*] Измените алгоритм E (из соображений эффективности) таким образом, чтобы исключить из него все тривиальные операции замены типа "$m \leftarrow n$". Запишите этот новый алгоритм в стиле алгоритма E и назовите его алгоритмом F.

4. [*16*] Чему равен наибольший общий делитель чисел 2 166 и 6 099?

▶ **5.** [*12*] Покажите, что для процедуры чтения книг этой серии, приведенной в предисловии, не хватает трех из пяти условий для того, чтобы она стала настоящим алгоритмом! Укажите также некоторые различия в форме записи этой процедуры и алгоритма E.

6. [*20*] Чему равно T_5 (среднее число случаев выполнения шага E1 при $n = 5$)?

▶ **7.** [*M21*] Пусть m известно, а n — любое целое положительное число. Пусть U_m — среднее число случаев выполнения шага E1 из алгоритма E. Покажите, что U_m четко определено. Существует ли какая-либо связь между U_m и T_m?

8. [*M25*] Придумайте эффективный формальный алгоритм вычисления наибольшего общего делителя целых положительных чисел m и n, определив соответствующим образом θ_j, ϕ_j, a_j, b_j (как в формулах (3)). Пусть входные данные представлены строкой $a^m b^n$, т. е. за a, взятым m раз, следует b, взятое n раз. Постарайтесь найти самое простое решение, насколько это возможно. [*Указание.* Воспользуйтесь алгоритмом E, но вместо деления на шаге E1 присвойте $r \leftarrow |m - n|$, $n \leftarrow \min(m, n)$.]

▶ **9.** [*M30*] Предположим, что $C_1 = (Q_1, I_1, \Omega_1, f_1)$ и $C_2 = (Q_2, I_2, \Omega_2, f_2)$ — методы вычислений. Например, C_1 может обозначать алгоритм E (см. формулу (2)) при условии, что m и n ограничены по величине, а C_2 — компьютерную программу, реализующую алгоритм E. (Тогда можно считать, что Q_2 — это набор всех состояний машины, т. е. всех возможных конфигураций ее памяти и регистров, f_2 определяет элементарную машинную операцию, а I_2 — начальное состояние, которое включает программу определения наибольшего общего делителя, а также значения m и n.)

Сформулируйте теоретико-множественное определение понятия "C_2 является представлением C_1" или "C_2 имитирует C_1". Интуитивно это означает, что любая вычисляемая последовательность C_1 имитируется C_2, за исключением того, что у C_2 может быть больше шагов, на которых выполняются вычисления, и можно получить больше информации из состояний. (Таким образом, мы получим точную интерпретацию утверждения "Программа X является реализацией алгоритма Y".)

1.2. МАТЕМАТИЧЕСКОЕ ВВЕДЕНИЕ

В ЭТОМ РАЗДЕЛЕ мы рассмотрим математические обозначения, используемые в книге *Искусство программирования*, и выведем основные формулы, которые будут часто применяться. Даже читатель, которого не интересуют сложные математические выкладки, должен понять *смысл* формул, чтобы иметь возможность пользоваться готовыми результатами.

Математические обозначения используются в этой книге для двух основных целей: для описания частей алгоритма и для анализа его рабочих характеристик. В предыдущем разделе были приведены обозначения, используемые в описаниях алгоритмов; как вы уже знаете, они достаточно просты. Но для анализа алгоритмов нам нужны другие, специальные обозначения.

Большинство рассматриваемых нами алгоритмов будет сопровождаться математическими подсчетами, определяющими ожидаемую скорость выполнения алгоритма. В этих вычислениях будут использоваться знания практически из всех разделов математики (для изложения всех используемых математических понятий потребовалась бы отдельная книга). Тем не менее для выполнения большинства вычислений используются знания математики на уровне школьного курса алгебры, поэтому читатель, знакомый с элементарными вычислениями, сможет разобраться почти во всех математических выкладках. В случаях, когда нам понадобятся более глубокие результаты из теории комплексного переменного, теории групп, теории чисел, теории вероятностей и т. д., материал будет излагаться как можно проще либо будет дана ссылка на другие источники информации.

Математические методы, используемые при анализе алгоритмов, имеют свои отличительные особенности. Например, нам довольно часто придется выполнять суммирование конечного числа рациональных чисел или решать рекуррентные уравнения. Подобные темы обычно очень поверхностно освещаются при чтении математических дисциплин, поэтому назначение следующих разделов — не только потренироваться в использовании обозначений, но и проиллюстрировать типы и методы вычислений, которые будут нам особенно необходимы.

Важное замечание. Хотя в следующих разделах содержатся обширные сведения из различных областей математики, которые совершенно необходимы для изучения компьютерных алгоритмов, большинство читателей сначала не увидят особой связи между этим материалом и программированием (за исключением раздела 1.2.1, в котором такая связь очевидна). Читатель, конечно, может сразу приступить к внимательному изучению следующих разделов, приняв на веру слова автора о том, что данные темы действительно очень важны. Но я считаю, что главной побудительной силой является интерес. Поэтому, наверное, лучше поступить иначе: *сначала просто просмотреть этот раздел, а затем* (после знакомства с применением указанных методов в следующих главах) *снова вернуться к нему для более глубокого изучения.* Ведь если во время первого чтения книги потратить слишком много времени на изучение данного математического материала, то можно никогда не дойти до вопросов программирования! Тем не менее каждый читатель должен ознакомиться хотя бы с общим содержанием этих разделов и даже во время первого чтения попытаться выполнить несколько упражнений. Разделу 1.2.10 следует уделить особое внимание, так как это отправная точка для большей части теоре-

тического материала, излагаемого впоследствии. В разделе 1.3 происходит резкий переход от "чистой математики" к "чистому программированию."

Более подробно последующий материал излагается в книге R. Graham, D. Knuth, O. Patashnik, *Concrete Mathematics*, second edition (Reading, Mass.: Addison-Wesley, 1994) (Грэхем Р., Кнут Д., Паташник О. *Конкретная математика.* — М.: Мир, 1998). Впоследствии, ссылаясь на эту книгу, мы будем называть ее просто *CMath*.

1.2.1. Математическая индукция

Пусть $P(n)$ — некоторое утверждение, касающееся целого числа n, например, "n умножить на $(n+3)$ — четное число" или "если $n \geq 10$, то $2^n > n^3$". Предположим, нам нужно *доказать, что утверждение $P(n)$ верно для всех положительных целых чисел n.* Существует важный метод доказательства этого факта, который состоит в следующем.

a) Доказать, что $P(1)$ верно.

b) Доказать, что "если $P(1), P(2), \ldots, P(n)$ справедливы, то $P(n+1)$ также справедливо"; это доказательство должно иметь силу для любого целого положительного n.

В качестве примера рассмотрим следующие известные с древних времен равенства, которые многие исследователи открывали независимо друг от друга:

$$\begin{aligned} 1 &= 1^2, \\ 1+3 &= 2^2, \\ 1+3+5 &= 3^2, \\ 1+3+5+7 &= 4^2, \\ 1+3+5+7+9 &= 5^2. \end{aligned} \tag{1}$$

В общем виде эти равенства можно записать следующим образом:

$$1+3+\cdots+(2n-1) = n^2. \tag{2}$$

Давайте назовем это утверждение $P(n)$ и докажем, что оно верно для любого положительного n. Согласно методу, описанному выше, имеем следующее.

a) "$P(1)$ верно, так как $1 = 1^2$."

b) "Если все утверждения $P(1), \ldots, P(n)$ справедливы, то, в частности, верно и $P(n)$; следовательно, выполняется соотношение (2). Добавляя к обеим частям этого уравнения $2n+1$, получаем

$$1+3+\cdots+(2n-1)+(2n+1) = n^2+2n+1 = (n+1)^2.$$

Таким образом, утверждение $P(n+1)$ также справедливо."

Этот метод можно считать *алгоритмической процедурой доказательства.* В самом деле, следующий алгоритм дает доказательство утверждения $P(n)$ для любого целого положительного n в предположении, что пп. (a) и (b) уже выполнены.

Алгоритм I (*Построить доказательство*). Для заданного целого положительного числа n этот алгоритм (рис. 2) выдаст доказательство того, что утверждение $P(n)$ верно.

I1. [Доказать $P(1)$.] Присвоить $k \leftarrow 1$ и в соответствии с п. (a) выдать доказательство утверждения $P(1)$.

I2. [$k = n$?] Если $k = n$, закончить выполнение алгоритма; требуемое доказательство выдано.

I3. [Доказать $P(k+1)$.] Согласно п. (b) выдать доказательство того, что "Если все утверждения $P(1), \ldots, P(k)$ справедливы, то $P(k+1)$ также справедливо". Вывести фразу "Мы уже доказали, что если утверждения $P(1), \ldots, P(k)$ верны, то верно и $P(k+1)$".

I4. [Увеличить k.] Увеличить k на 1 и перейти к шагу I2. ▮

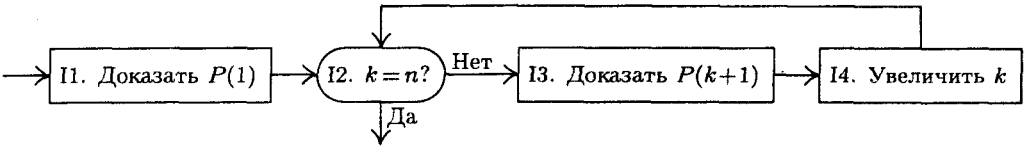


Рис. 2. Алгоритм I: математическая индукция.

Поскольку этот алгоритм выдает доказательство утверждения $P(n)$ для любого заданного n, метод доказательства, сформулированный в пп. (a) и (b), логически обоснован. Он называется *доказательством методом математической индукции.*

Понятие математической индукции следует отличать от того, что в научной практике обычно называют индуктивным методом. Данный метод заключается в том, что ученый делает некоторые наблюдения и создает "по индукции" общую теорию или выдвигает гипотезу, объясняющую эти факты. Например, на основании пяти соотношений (1), приведенных выше, мы могли бы сформулировать соотношение (2). В этом смысле индукция — не более чем догадка или попытка объяснить конкретную ситуацию; математики называют это эмпирическим результатом или предположением.

Для того чтобы прояснить суть дела, рассмотрим еще один поучительный пример. Пусть $p(n)$ обозначает количество *разбиений* числа n, т. е. количество различных способов записи числа n в виде суммы целых положительных чисел (порядок слагаемых значения не имеет). Так как для числа 5 существует ровно семь таких способов записи, т. е.

$$1+1+1+1+1 = 2+1+1+1 = 2+2+1 = 3+1+1 = 3+2 = 4+1 = 5,$$

то $p(5) = 7$. На самом деле установить первые пять значений функции $p(n)$ довольно легко:

$$p(1) = 1, \quad p(2) = 2, \quad p(3) = 3, \quad p(4) = 5, \quad p(5) = 7.$$

На этом основании мы могли бы предварительно сформулировать по индукции предположение о том, что последовательность $p(2)$, $p(3)$, ... пробегает множество

простых чисел. Для проверки данной гипотезы продолжаем вычисления и находим $p(6)$. Ура! $p(6) = 11$, что подтверждает наше предположение.

[Но, к сожалению, оказывается, что $p(7)$ равно 15. Увы, все идет насмарку, и приходится начинать сначала. Известно, что значения $p(n)$ отличаются довольно сложным поведением, хотя С. Рамануджану (S. Ramanujan) удалось угадать и доказать много замечательных фактов, касающихся этих чисел. Более подробную информацию можно найти в книге G. H. Hardy, *Ramanujan* (London: Cambridge University Press, 1940), гл. 6 и 8.]

Математическая индукция не имеет ничего общего с тем индуктивным методом, который мы только что описали. Это не догадка, а неопровержимое доказательство утверждения; скажу больше: это доказательство бесконечного числа утверждений, по одному для каждого n. "Индукцией" этот метод назван только потому, что сначала нужно выдвинуть предположение о том, *что* нужно доказать, а *затем* уже применять метод математической индукции. Начиная с этого момента, слово "индукция" в книге будет использоваться только для обозначения доказательства методом математической индукции.

Существует геометрический способ доказательства соотношения (2). На рис. 3 для $n = 6$ показано, что n^2 клеток разбиты на группы $1 + 3 + \cdots + (2n - 1)$ клеток. Но в конечном счете этот рисунок можно считать "доказательством" только в случае, если мы покажем, что данное построение можно выполнить для любого n. А это, в сущности, и будет доказательством по индукции.

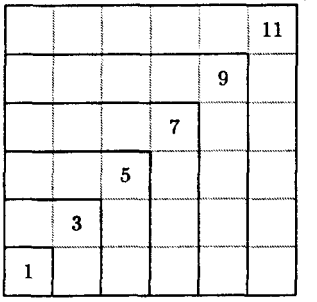


Рис. 3. Нечетные числа в сумме дают квадрат.

В нашем доказательстве соотношения (2) был использован только частный случай (b); мы просто показали, что из справедливости $P(n)$ следует справедливость $P(n + 1)$. Это очень важный случай, который встречается довольно часто, но в следующем примере будут проиллюстрированы более широкие возможности метода математической индукции. Определим *последовательность Фибоначчи* F_0, F_1, F_2, ... с помощью такого правила: $F_0 = 0$, $F_1 = 1$, а каждый последующий член равен сумме двух предыдущих. Таким образом, первые члены этой последовательности выглядят так: 0, 1, 1, 2, 3, 5, 8, 13, ... (более подробно мы изучим эту последовательность в разделе 1.2.8). А теперь докажем, что если через ϕ обозначить число $(1 + \sqrt{5})/2$, то имеем

$$F_n \le \phi^{n-1} \tag{3}$$

для всех целых положительных n. Назовем эту формулу утверждением $P(n)$.

Если $n = 1$, то $F_1 = 1 = \phi^0 = \phi^{n-1}$, поэтому п. (a) выполнен. Переходя к п. (b), заметим сначала, что $P(2)$ также справедливо, поскольку $F_2 = 1 < 1.6 < \phi^1 = \phi^{2-1}$. А теперь, если все $P(1)$, $P(2)$, ..., $P(n)$ справедливы и $n > 1$, то мы знаем, в частности, что справедливы $P(n - 1)$ и $P(n)$. Поэтому $F_{n-1} \le \phi^{n-2}$ и $F_n \le \phi^{n-1}$ Складывая данные неравенства, получаем

$$F_{n+1} = F_{n-1} + F_n \le \phi^{n-2} + \phi^{n-1} = \phi^{n-2}(1 + \phi). \tag{4}$$

Важное свойство числа ϕ, которое и является причиной первоочередного выбора этого числа для данной задачи, состоит в том, что

$$1 + \phi = \phi^2. \tag{5}$$

Подставив (5) в (4), получим $F_{n+1} \le \phi^n$, а это и есть утверждение $P(n+1)$. Таким образом, п. (b) выполнен и формула (3) доказана методом математической индукции. Обратите внимание, что п. (b) мы выполняли двумя различными способами: *непосредственно* доказали $P(n+1)$ при $n = 1$ и использовали индуктивный метод при $n > 1$. Это было необходимо, так как при $n = 1$ ссылка на $P(n-1) = P(0)$ была бы незаконной.

Математическую индукцию можно использовать также для доказательства фактов, касающихся *алгоритмов*. Давайте рассмотрим следующее обобщение алгоритма Евклида.

Алгоритм E (*Обобщенный алгоритм Евклида*). Даны два целых положительных числа m и n. Требуется найти их наибольший общий делитель* d и два целых числа a и b, таких, что $am + bn = d$.

E1. [Инициализация.] Присвоить $a' \leftarrow b \leftarrow 1$, $a \leftarrow b' \leftarrow 0$, $c \leftarrow m$, $d \leftarrow n$.

E2. [Деление.] Пусть q и r — это частное и остаток от деления c на d соответственно. (Тогда $c = qd + r$, где $0 \le r < d$.)

E3. [Остаток — нуль?] Если $r = 0$, то выполнение алгоритма прекращается; в этом случае имеем $am + bn = d$, как и требовалось.

E4. [Повторение цикла.] Присвоить $c \leftarrow d$, $d \leftarrow r$, $t \leftarrow a'$, $a' \leftarrow a$, $a \leftarrow t - qa$, $t \leftarrow b'$, $b' \leftarrow b$, $b \leftarrow t - qb$ и вернуться к шагу E2. ∎

Если изъять из алгоритма переменные a, b, a' и b' и использовать m и n в качестве вспомогательных переменных c и d, то получим старый алгоритм 1.1E. В новой версии алгоритма выполняется немного больше вычислений, так как необходимо определить коэффициенты a и b. Предположим, что $m = 1\,769$ и $n = 551$. Тогда последовательно (после шага E2) имеем:

a'	a	b'	b	c	d	q	r
1	0	0	1	1769	551	3	116
0	1	1	−3	551	116	4	87
1	−4	−3	13	116	87	1	29
−4	5	13	−16	87	29	3	0

Проверяя полученные результаты, убеждаемся в том, что все правильно, так как $5 \times 1769 - 16 \times 551 = 8845 - 8816 = 29$, т. е. мы получили наибольший общий делитель чисел 1 769 и 551.

Теперь нужно *доказать*, что рассматриваемый алгоритм работает правильно при любых m и n. Для этого попробуем применить метод математической индукции к следующему утверждению $P(n)$: "Алгоритм E дает правильное решение для заданного n и всех целых m". Но провести подобное доказательство не так-то просто, поэтому нужно доказать сначала несколько дополнительных фактов. После

* Сокращенно — gcd (greatest common divisor). — *Прим. перев.*

некоторого изучения проблемы выясняется, что нужно доказать какой-то факт, связанный с коэффициентами a, b, a' и b'. Этот факт заключается в том, что равенства

$$a'm + b'n = c, \qquad am + bn = d \tag{6}$$

верны в каждом случае выполнения шага E2. Данные равенства можно доказать непосредственно, заметив, что они безусловно справедливы после первого выполнения шага E2 и что шаг E4 не меняет это положение вещей (см. упр. 6).

Теперь мы готовы индукцией по n доказать, что алгоритм E работает правильно. Если m кратно n, то очевидно, что алгоритм работает правильно, поскольку его работа заканчивается на шаге E3 в первом же цикле и мы получаем верный результат. Это происходит всегда, когда $n = 1$. Поэтому остается провести доказательство для случая, когда $n > 1$ и m не является кратным n. В такой ситуации в первом цикле осуществляется переход к шагу E4 и выполняются операции присвоения $c \leftarrow n$, $d \leftarrow r$. И так как $r < n$, *по индукции можно предположить*, что окончательное значение d — это наибольший общий делитель чисел n и r. Из доказательства, приведенного в разделе 1.1, следует, что пары $\{m, n\}$ и $\{n, r\}$ имеют одинаковые наибольшие общие делители и, в частности, один и тот же наибольший общий делитель. Значит, d — это наибольший общий делитель чисел m и n и согласно (6) $am + bn = d$.

Фраза, которая в приведенном выше доказательстве выделена курсивом, является иллюстрацией того общепринятого условного языка, который так часто используется в доказательствах методом индукции. Например, выполняя п. (b), вместо того чтобы сказать "Теперь предположим, что утверждения $P(1), P(2), \ldots, P(n)$ справедливы, и на этом основании докажем справедливость утверждения $P(n+1)$", мы будем говорить просто "Теперь докажем утверждение $P(n)$; *по индукции мы можем предположить*, что $P(k)$ верно для любого $1 \le k < n$".

Если хорошо вдуматься и посмотреть на все вышесказанное с несколько иной точки зрения, то перед нами предстанет общий метод доказательства корректности *любого* алгоритма. Идея состоит в том, чтобы взять блок-схему некоторого алгоритма и к каждой стрелке добавить примечание о текущем состоянии дел в тот момент, который соответствует стрелке. На рис. 4 эти примечания (мы их будем называть также утверждениями) обозначены через *A1*, *A2*, ..., *A6*. (Все они сделаны при дополнительном условии, что переменные принимают только целые значения; это условие опущено с целью экономии места.) В утверждении *A1* даются первоначальные предположения о входных данных алгоритма, а в *A4* формулируется положение о том, что мы хотим доказать по поводу выходных значений a, b и d.

Общий метод заключается в том, чтобы для каждого блока на блок-схеме доказать следующее:

> *если любое утверждение, которое соответствует стрелке, ведущей к блоку, верно до выполнения операции из этого блока, то все утверждения, которые соответствуют стрелкам, ведущим от блока, также верны после выполнения операции.* (7)

Согласно описанному методу для нашего примера мы должны доказать, что если до выполнения шага E2 верно *A2* либо *A6*, то после выполнения этого шага верно *A3*. (В данном случае утверждение *A2* является более сильным, чем *A6*, т. е. из *A2*

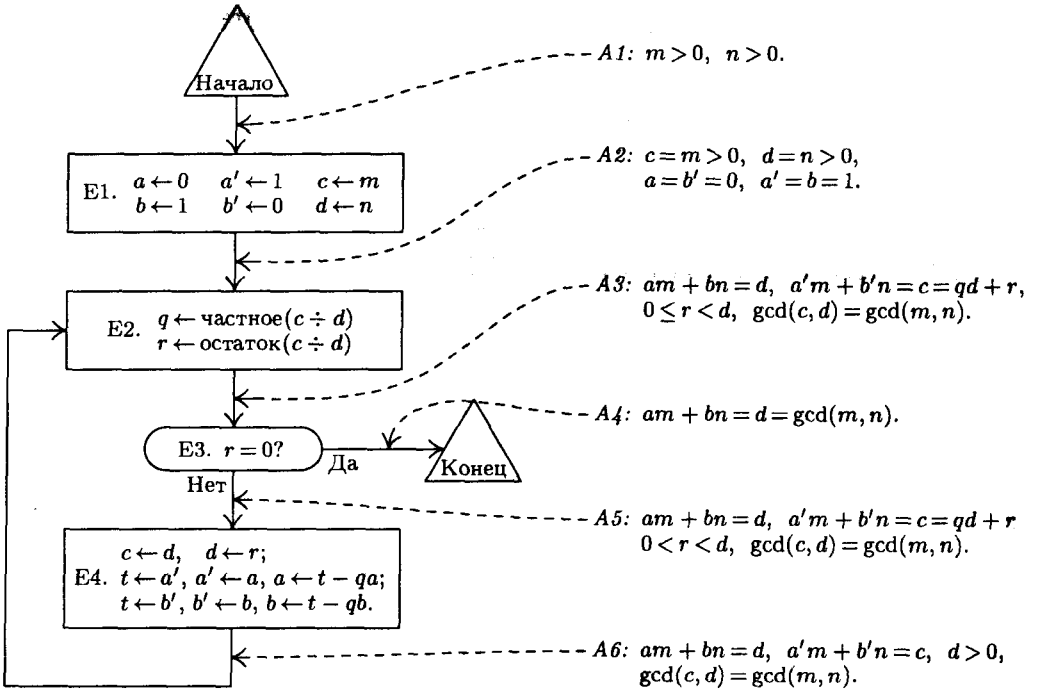


Рис. 4. Блок-схема алгоритма E, дополненная примечаниями, которые доказывают правильность работы алгоритма.

следует *A6*. Поэтому нам достаточно доказать, что выполнение *A6* до шага E2 влечет за собой выполнение *A3* после этого шага. Заметим, что условие $d > 0$ необходимо в *A6* для того, чтобы операция E2 имела смысл.) Нужно показать также, что из *A3* (при условии, что $r = 0$) следует *A4*, из *A3* (при условии, что $r \neq 0$) следует *A5* и т. д. Все это доказывается очень просто.

Если доказать утверждение (7) *для каждого блока, то все примечания к стрелкам будут верны в любом случае выполнения алгоритма.* Теперь мы можем применить индукцию по числу шагов, т. е. по числу стрелок в блок-схеме. При прохождении первой стрелки (той, которая выходит из блока "Начало") утверждение *A1* верно, поскольку мы всегда исходим из предположения, что входные значения удовлетворяют заданным условиям. Таким образом, утверждение, которое соответствует первой стрелке, верно. Если утверждение, которое соответствует n-й стрелке, верно, то согласно (7) утверждение, которое соответствует $(n + 1)$-й стрелке, тоже верно.

Исходя из этого общего метода доказательство правильности заданного алгоритма, очевидно, сводится к нахождению правильных утверждений, соответствующих стрелкам блок-схемы. Как только данное начальное препятствие будет преодолено, останется лишь рутинная работа, связанная с доказательством того, что каждое утверждение на входе в блок влечет за собой утверждение на выходе из блока. В действительности после того как вы придумаете самые трудные из этих утверждений, найти все остальные уже не составит труда. Скажем, если даны утверждения *A1*, *A4* и *A6*, уже понятно, какими должны быть утверждения *A2*, *A3* и *A5*. В нашем

примере самых больших творческих усилий потребует доказательство утверждения *A6*; все остальное, в принципе, должно получиться автоматически. Поэтому я и не пытался давать для алгоритмов, приведенных в книге, формальные доказательства с той степенью детализации, которая отражена на рис. 4. Вполне достаточно сформулировать только главные утверждения. Обычно они приводятся либо в ходе обсуждения алгоритма, либо даются в скобках в тексте самого алгоритма.

Этот подход к доказательству корректности алгоритма имеет и другой, еще более важный аспект: *он отражает способ нашего понимания алгоритма.* Если помните, в разделе 1.1 я предупреждал о том, чтобы вы не читали алгоритмы, как роман. Я рекомендую проверять работу алгоритма на примере одного-двух наборов входных данных. И это не случайно, так как пробная "прогонка" алгоритма поможет вам мысленно сформулировать утверждения, соответствующие стрелкам на блок-схеме. Автор твердо убежден, что истинная уверенность в корректности алгоритма приходит только тогда, когда мысленно сформулированы все утверждения, приведенные на рис. 4. Отсюда следуют важные психологические выводы, касающиеся передачи алгоритма от одного лица к другому. Речь идет о том, что, объясняя алгоритм кому-либо другому, всегда следует явно формулировать основные утверждения, которые трудно получить автоматически. Например, в случае алгоритма E нужно обязательно упомянуть утверждение *A6*.

Но бдительный читатель, конечно, заметил зияющую брешь в нашем последнем доказательстве алгоритма E. Из него нигде не следует, что алгоритм обладает свойством конечности, т. е. рано или поздно его выполнение завершится. Мы доказали только, что *если* алгоритм конечен, то он дает правильный результат!

(Например, заметим, что алгоритм E по-прежнему имеет смысл, если его переменные m, n, c, d и r принимают значения типа $u + v\sqrt{2}$, где u и v — целые числа*. Переменные q, a, b, a', b' должны по-прежнему принимать целые значения. Если, например, на вход подать значения $m = 12-6\sqrt{2}$ и $n = 20-10\sqrt{2}$, то на выходе будет получен "наибольший общий делитель" $d = 4-2\sqrt{2}$ и коэффициенты $a = +2$, $b = -1$. Даже при таком расширении исходных предположений доказательства утверждений от *A1* до *A6* остаются в силе. Следовательно, на любом этапе выполнения этой процедуры все утверждения верны. Но если начать со значений $m = 1$ и $n = \sqrt{2}$, то вычисления никогда не закончатся (см. упр. 12). Следовательно, из доказательства утверждений *A1–A6* еще *не* следует, что алгоритм конечен.)

Доказательства конечности алгоритмов обычно проводят отдельно. Но в упр. 13 показано, что во многих важных случаях приведенный выше метод *можно* обобщить таким образом, чтобы включить доказательство конечности в виде промежуточного результата.

Итак, мы уже дважды доказали правильность алгоритма E. Чтобы быть последовательными до конца, нам следовало бы попытаться доказать, что первый алгоритм в этом разделе, а именно — алгоритм I, также корректен. Ведь, в сущности, мы использовали алгоритм I, чтобы показать корректность любого доказательства по индукции. Но если мы попытаемся *доказать*, что алгоритм I работает правильно, то попадем в затруднительное положение: мы не сможем сделать это, не воспользовавшись снова индукцией! Итак, получается замкнутый круг.

* Определение деления с остатком в этом случае приведено в решении к упр. 12. — *Прим. ред.*

В последнее время *любое* свойство целых чисел принято доказывать с помощью индукции в ту или иную сторону. Ведь если мы обратимся к основным понятиям, то увидим, что целые числа, в сущности, *определяются* по индукции. Поэтому можно принять в качестве аксиомы утверждение о том, что любое целое положительное число n либо равно 1, либо может быть получено, если взять 1 за исходное значение и последовательно прибавлять по единице. Этого достаточно, чтобы доказать правильность алгоритма I. [Более подробно фундаментальные понятия, связанные с целыми числами, рассматриваются в статье Leon Henkin, On Mathematical Induction, *AMM* **67** (1960), 323–338.]

Таким образом, метод математической индукции глубоко связан с понятием числа. Первым европейцем, применившим в 1575 году метод математической индукции для получения строгих доказательств, был итальянский ученый Франческо Мауролико (Francesco Maurolico). В начале 17 века Пьер де Ферма (Pierre de Fermat) усовершенствовал этот метод; он называл его методом бесконечного спуска. Это понятие явно используется также в последних трудах Блеза Паскаля (Blaise Pascal) (1653). Термин "математическая индукция", видимо, был придуман А. Де Морганом (A. De Morgan) в начале 19 века. [См. *AMM* **24** (1917), 199–207; **25** (1918), 197–201; *Arch. Hist. Exact Sci.* **9** (1972), 1–21.] Более подробно метод математической индукции рассматривается в книге Д. Пойа (G. Pólya) *Induction and Analogy in Mathematics* (Princeton, N. J.: Princeton University Press, 1954), Chapter 7 ("Математика и правдоподобные рассуждения"; т. 1, "Индукция и аналогия в математике" (М.: Изд-во иностр. лит., 1957), гл. 7).

Описанный выше метод доказательства алгоритмов с помощью утверждений, соответствующих стрелкам, и индукции, по существу, принадлежит Р. В. Флойду (R. W. Floyd). Он показал, что смысловое определение каждой операции в языке программирования можно сформулировать в виде логического правила. Это правило точно определяет, какие утверждения могут быть верны после выполнения операции, если известно, какие утверждения верны до ее выполнения. [См. "Assigning Meanings to Programs", *Proc. Symp. Appl. Math.*, Amer. Math. Soc., **19** (1967), 19–32.] Аналогичные идеи были независимо высказаны Питером Науром (Peter Naur), *BIT* **6** (1966), 310–316, который называл утверждения, соответствующие стрелкам на блок-схемах, общими снимками (general snapshots). Необходимо уточнить, что понятие "инвариант" было введено Ч. Э. Р. Хоаром (C. A. R. Hoare) (например, см. *CACM* **14** (1971), 39–45). В более поздних публикациях считалось, что выгоднее изменить направление, заданное Флойдом, на противоположное, т. е. что нужно исходить из утверждения, которое должно выполняться *после* операции, и доказывать, что "самое слабое предусловие", которое должно иметь место *до* выполнения этой операции, на самом деле имеет место. Подобный подход позволяет разрабатывать новые алгоритмы, выбирая в качестве отправной точки характеристики желаемых выходных данных и двигаясь в обратном направлении (т. е. вверх по блок-схеме). При выполнении этого условия полученные алгоритмы обязательно будут корректными. [См. E. W. Dijkstra, *CACM* **18** (1975), 453–457; *A Discipline of Programming* (Prentice-Hall, 1976).]

На самом деле зачатки идеи индуктивных утверждений появились в 1946 году, в то время, когда Г. Г. Голдстайн (H. H. Goldstine) и Дж. фон Нейман (J. von Neumann) изобрели блок-схемы. Их первоначальные блок-схемы включали в себя "блоки с утверждениями", которые очень похожи на утверждения, показанные на рис. 4.

[См. John von Neumann, *Collected Works* **5** (New York: Macmillan, 1963), 91–99. См. также ранние комментарии А. М. Тьюринга (A. M. Turing) о проверке корректности алгоритмов в *Report of a Conference on High Speed Automatic Calculating Machines* (Cambridge Univ., 1949), 67–68 и рисунки. Эта работа переиздана с комментариями Ф. Л. Моррис (F. L. Morris) и К. Б. Джонс (C. B. Jones) в *Annals of the History of Computing* **6** (1984), 139–143.]

В понимании теории программ
большую помощь могут оказать один-два оператора,
описывающих состояние машины в тщательно выбранные
моменты времени ...
Высшая форма теоретического метода — это
предоставление для утверждений неопровержимых математических доказательств.
А высшая форма экспериментального метода — это
тестирование программы на машине для различных начальных условий
и объявление ее годной, если утверждения выполняются в каждом случае.
Каждый из методов имеет свои недостатки.

— А. М. ТЬЮРИНГ (A. M. TURING),
Ferranti Mark I Programming Manual (1950)

УПРАЖНЕНИЯ

1. [*05*] Объясните, как можно модифицировать идею доказательства методом математической индукции в случае, если некоторое утверждение $P(n)$ нужно доказать для всех *неотрицательных* целых чисел, т. е. для $n = 0, 1, 2, \ldots$, а не для $n = 1, 2, 3, \ldots$.

2. [*15*] Найдите ошибку в следующем доказательстве. "**Теорема.** Пусть a — любое положительное число. Для всех целых положительных чисел n имеем $a^{n-1} = 1$. *Доказательство.* Если $n = 1$, $a^{n-1} = a^{1-1} = a^0 = 1$. По индукции, предполагая, что теорема верна для $1, 2, \ldots, n$, имеем

$$a^{(n+1)-1} = a^n = \frac{a^{n-1} \times a^{n-1}}{a^{(n-1)-1}} = \frac{1 \times 1}{1} = 1;$$

следовательно, теорема верна также для $n + 1$."

3. [*18*] Следующее доказательство по индукции выглядит корректным, но по непонятной причине для $n = 6$ левая часть уравнения дает $\frac{1}{2}+\frac{1}{6}+\frac{1}{12}+\frac{1}{20}+\frac{1}{30} = \frac{5}{6}$, а правая — $\frac{3}{2}-\frac{1}{6} = \frac{4}{3}$. В чем же ошибка? "**Теорема.**

$$\frac{1}{1 \times 2} + \frac{1}{2 \times 3} + \cdots + \frac{1}{(n-1) \times n} = \frac{3}{2} - \frac{1}{n}.$$

Доказательство. Используем индукцию по n. Для $n = 1$ доказательство очевидно: $3/2 - 1/n = 1/(1 \times 2)$. Предполагая, что теорема верна для n, имеем:

$$\frac{1}{1 \times 2} + \cdots + \frac{1}{(n-1) \times n} + \frac{1}{n \times (n+1)}$$
$$= \frac{3}{2} - \frac{1}{n} + \frac{1}{n(n+1)} = \frac{3}{2} - \frac{1}{n} + \left(\frac{1}{n} - \frac{1}{n+1}\right) = \frac{3}{2} - \frac{1}{n+1}."$$

4. [*20*] Докажите, что числа Фибоначчи удовлетворяют не только соотношению (3), но и неравенству $F_n \geq \phi^{n-2}$.

5. [*21*] *Простое число* — это целое число, большее единицы, которое делится только на 1 и на само себя. Используя данное определение и метод математической индукции, докажите, что любое целое число, большее единицы, можно записать как произведение одного или нескольких простых чисел. (Для удобства будем считать, что простое число — это "произведение" одного простого числа, т. е. его самого.)

6. [*20*] Докажите, что если соотношения (6) справедливы непосредственно перед выполнением шага E4, то они верны и после его выполнения.

7. [*23*] Сформулируйте и докажите по индукции правило вычисления сумм 1^2, $2^2 - 1^2$, $3^2 - 2^2 + 1^2$, $4^2 - 3^2 + 2^2 - 1^2$, $5^2 - 4^2 + 3^2 - 2^2 + 1^2$ и т. д.

▸ **8.** [*25*] (a) Докажите по индукции следующую теорему Никомаха (Nicomachus) (ок. 100 г. н. э.): $1^3 = 1$, $2^3 = 3+5$, $3^3 = 7+9+11$, $4^3 = 13+15+17+19$ и т. д.* (b) Воспользуйтесь этим результатом для доказательства замечательной формулы $1^3+2^3+\cdots+n^3 = (1+2+\cdots+n)^2$.

[*Замечание.* Интересная геометрическая интерпретация этой формулы, предложенная автору P. B. Флойдом, показана на рис. 5. Идея этого построения связана с теоремой Никомаха и рис. 3. Другие "наглядные" доказательства можно найти в книгах Мартина Гарднера (Martin Gardner), *Knotted Doughnuts* (New York: Freeman, 1986), Chapter 16, а также J. H. Conway, R. K. Guy, *The Book of Numbers* (New York: Copernicus, 1996), Chapter 2.]

$$\begin{aligned}\text{Сторона} &= 5+5+5+5+5+5 = 5\cdot(5+1)\\ \text{Сторона} &= 5+4+3+2+1+1+2+3+4+5\\ &= 2(1+2+\cdots+5)\\ \text{Площадь} &= 4\cdot 1^2+4\cdot 2\cdot 2^2+4\cdot 3\cdot 3^2+4\cdot 4\cdot 4^2+4\cdot 5\cdot 5^2\\ &= 4(1^3+2^3+\cdots+5^3)\end{aligned}$$

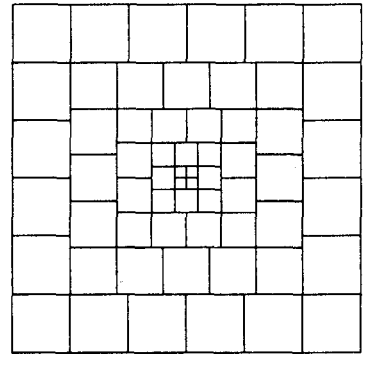


Рис. 5. Геометрическая интерпретация упр. 8, (b) при $n = 5$.

9. [*20*] Докажите по индукции, что если $0 < a < 1$, то $(1-a)^n \geq 1 - na$.

10. [*M22*] Докажите по индукции, что если $n \geq 10$, то $2^n > n^3$.

11. [*M30*] Найдите и докажите простую формулу для следующей суммы:

$$\frac{1^3}{1^4+4} - \frac{3^3}{3^4+4} + \frac{5^3}{5^4+4} - \cdots + \frac{(-1)^n(2n+1)^3}{(2n+1)^4+4}.$$

12. [*M25*] Покажите, как можно обобщить алгоритм E, чтобы, как было указано в тексте, для него допускались входные значения вида $u+v\sqrt{2}$, где u и v — целые числа, и вычисления по-прежнему выполнялись элементарным образом (т. е. не выражая иррациональное число $\sqrt{2}$ бесконечной десятичной дробью). Докажите, что при $m = 1$ и $n = \sqrt{2}$ выполнение алгоритма никогда не закончится.

▸ **13.** [*M23*] Обобщите алгоритм E, введя новую переменную T и добавив в начале каждого шага операцию $T \leftarrow T+1$. (Таким образом, переменная T — счетчик выполненных шагов.)

---

* Требуется доказать формулу $(n^2 - n + 1) + (n^2 - n + 3) + \cdots + (n^2 + n - 1) = n^3$. — *Прим. перев.*

Предположим, что первоначальное значение T равно нулю, поэтому утверждение *A1* на рис. 4 примет вид $m > 0$, $n > 0$, $T = 0$. Аналогично к *A2* следует добавить дополнительное условие $T = 1$. Покажите, как добавить к утверждениям дополнительные условия таким образом, чтобы из любого утверждения *A1*, *A2*, ..., *A6* следовало, что $T \leq 3n$, и чтобы можно было провести доказательство по индукции. (Следовательно, вычисления должны закончиться максимум через $3n$ шагов.)

14. [*50*] (Р. В. Флойд.) Составьте компьютерную программу, на вход которой подаются программы, написанные на некотором языке программирования, вместе с необязательными утверждениями и которая пытается сформулировать остальные утверждения, необходимые для доказательства корректности входной компьютерной программы. Например, постарайтесь составить программу, с помощью которой можно доказать корректность алгоритма E, если даны только утверждения *A1*, *A4* и *A6*. Более подробную информацию об этом можно найти в статьях Р. В. Флойда и Дж. К. Кинга (J. C. King) (IFIP Congress proceedings, 1971).

▸ **15.** [*HM28*] (*Обобщенная индукция.*) В тексте показано, как доказывать по индукции утверждения $P(n)$, зависящие от единственного целого числа n, но ничего не говорится о том, как доказывать утверждения $P(m, n)$, зависящие от двух целых чисел. В подобных случаях обычно используют что-то вроде "двойной индукции", и поэтому доказательство часто выглядит довольно запутанным. Но на самом деле существует важный принцип доказательства, который является более общим, чем простая индукция, и применяется не только в подобных случаях, но и тогда, когда утверждения нужно доказывать для несчетных множеств, например если нужно доказать $P(x)$ для всех действительных x. Этот общий принцип называется *вполне упорядоченностью.*

Пусть "$\prec$" — отношение на множестве S, удовлетворяющее следующим свойствам.

i) Для любых x, y и z из S, если $x \prec y$ и $y \prec z$, то $x \prec z$.

ii) Для любых x и y из S выполняется ровно одна из следующих трех возможностей: $x \prec y$, $x = y$ либо $y \prec x$.

iii) Если A — любое непустое подмножество S, то существует элемент x из A, такой, что $x \preceq y$ (т. е. $x \prec y$ или $x = y$) для всех y из A.

Говорят, что это отношение вполне упорядочивает множество S. Например, очевидно, что множество положительных целых чисел вполне упорядочено обычным отношением "меньше чем" ("$<$").

a) Покажите, что множество *всех* целых чисел не является вполне упорядоченным отношением "$<$".

b) Определите на множестве всех целых чисел отношение вполне упорядоченности.

c) Является ли множество всех неотрицательных действительных чисел вполне упорядоченным отношением "$<$"?

d) (*Лексикографический порядок.*) Пусть множество S вполне упорядочено отношением "$\prec$" и пусть T_n, $n > 0$, — множество всех наборов $(x_1, x_2, \ldots, x_n)$ элементов x_j из S. Определим отношение $(x_1, x_2, \ldots, x_n) \prec (y_1, y_2, \ldots, y_n)$, если существует некоторое k, $1 \leq k \leq n$, такое, что $x_j = y_j$ для $1 \leq j < k$, а $x_k \prec y_k$ в S. Будет ли отношение "$\prec$" вполне упорядочивать T_n?

e) Продолжая п. (d), положим $T = \bigcup_{n \geq 1} T_n$; определим отношение $(x_1, x_2, \ldots, x_m) \prec (y_1, y_2, \ldots, y_n)$, если $x_j = y_j$ для $1 \leq j < k$ и $x_k \prec y_k$ для некоторого $k \leq \min(m, n)$ или если $m < n$ и $x_j = y_j$ для $1 \leq j \leq m$. Будет ли отношение "$\prec$" вполне упорядочивать T?

f) Покажите, что отношение "$\prec$" вполне упорядочивает множество S тогда и только тогда, когда оно удовлетворяет приведенным выше условиям (i) и (ii) и не существует бесконечной последовательности x_1, x_2, x_3, ..., такой, что $x_{j+1} \prec x_j$ для всех $j \geq 1$.

g) Пусть множество S вполне упорядочено отношением "$\prec$" и пусть $P(x)$ — это утверждение, зависящее от элемента x из S. Покажите, что если $P(x)$ можно доказать, предполагая, что $P(y)$ верно для всех $y \prec x$, то $P(x)$ верно для *всех* x из S.

[*Замечания.* П. (g) — это обещанное выше обобщение простой индукции. Если S — это множество положительных целых чисел, то мы получаем простой случай математической индукции, рассмотренный в тексте. Причем нас просят доказать, что $P(1)$ верно, если $P(y)$ верно для всех положительных целых $y < 1$; это то же самое, что просить нас просто доказать $P(1)$, поскольку $P(y)$ безусловно верно для всех таких y (таких y просто не существует). Итак, во многих случаях доказательство $P(1)$ не требует особой аргументации.

П. (d) в сочетании с (g) дает нам довольно сильный метод n-мерной индукции для доказательства утверждений $P(m_1, \ldots, m_n)$, зависящих от n целых положительных чисел $m_1, \ldots, m_n$.

П. (f) можно применить к компьютерным алгоритмам следующим образом. Если каждому состоянию вычисления x поставить в соответствие элемент $f(x)$, принадлежащий вполне упорядоченному множеству S, таким образом, чтобы каждый шаг вычислений переводил состояние x в состояние y так, что $f(y) \prec f(x)$, то алгоритм будет конечным. Этот принцип обобщает рассуждения о строго убывающих значениях n, которые использовались при доказательстве конечности алгоритма 1.1E.]

1.2.2. Числа, степени и логарифмы

Давайте начнем с того, что поближе познакомимся с числами, с которыми нам придется иметь дело. *Целые числа* — это

$$\ldots, -3, -2, -1, 0, 1, 2, 3, \ldots$$

(отрицательные, положительные и нуль). *Рациональное число* — это отношение (частное) двух целых чисел, p/q, где q — положительное число. *Действительное число* — это величина x, которая имеет *десятичное представление*:

$$x = n + 0.d_1d_2d_3\ldots, \tag{1}$$

где n — целое число, а каждое d_i — любая цифра от 0 до 9, причем в конце не должно быть бесконечной последовательности из идущих подряд девяток. Представление (1) означает, что для любого положительного k

$$n + \frac{d_1}{10} + \frac{d_2}{100} + \cdots + \frac{d_k}{10^k} \le x < n + \frac{d_1}{10} + \frac{d_2}{100} + \cdots + \frac{d_k}{10^k} + \frac{1}{10^k}. \tag{2}$$

Вот два примера действительных чисел, которые не являются рациональными:

$\pi = 3.14159265358979\ldots$, отношение длины окружности к ее диаметру;

$\phi = 1.61803398874989\ldots$, *золотое сечение* $(1 + \sqrt{5})/2$ (см. раздел 1.2.8).

Таблица важнейших числовых постоянных с точностью до сорокового десятичного знака приведена в приложении А. Думаю, нет необходимости обсуждать хорошо известные всем свойства сложения, вычитания, умножения, деления и сравнения действительных чисел.

Сложные проблемы, возникающие при оперировании целыми числами, часто решают с помощью действительных чисел, а сложные проблемы, связанные с действительными числами, — с помощью более общего класса величин, которые называются комплексными числами. *Комплексное число* — это величина z, которую

можно представить в виде $z = x + iy$, где x и y — действительные числа, а i — особая величина, удовлетворяющая уравнению $i^2 = -1$*. Будем называть x и y *действительной* и *мнимой* частями z и определим модуль комплексного числа z как

$$|z| = \sqrt{x^2 + y^2}. \tag{3}$$

Под комплексным числом, *сопряженным* к z, понимают $\overline{z} = x - iy$, и, таким образом, $z\overline{z} = x^2 + y^2 = |z|^2$. Теория комплексных чисел во многих отношениях проще и изящнее теории действительных чисел, хотя она и считается более сложной. Поэтому в данной книге мы ограничимся действительными числами, за исключением случаев, когда использование только действительных чисел приводит к необоснованным сложностям.

Если u и v — действительные числа, для которых $u \le v$, то *замкнутым интервалом* $[u\,..\,v]$ будем называть множество действительных чисел x, таких, что $u \le x \le v$. Аналогично *открытый интервал* $(u\,..\,v)$ — это множество действительных x, для которых $u < x < v$. И, наконец, *полуоткрытые интервалы* $[u\,..\,v)$ или $(u\,..\,v]$ определяются подобным образом. Мы допускаем также, что u может принимать значение $-\infty$, а v — ∞; в этом случае соответствующая сторона интервала остается открытой и мы считаем, что нижней или верхней границы у него нет. Таким образом, запись $(-\infty\,..\,\infty)$ обозначает множество всех действительных чисел, а запись $[0\,..\,\infty)$ — множество неотрицательных действительных чисел.

В дальнейшем в этом разделе буква b будет обозначать положительное действительное число. Если n — целое число, то b^n определяется известными правилами

$$b^0 = 1, \qquad b^n = b^{n-1}b, \quad \text{если} \quad n > 0, \qquad b^n = b^{n+1}/b, \quad \text{если} \quad n < 0. \tag{4}$$

По индукции легко доказать справедливость *правил возведения в степень*:

$$b^{x+y} = b^x b^y, \qquad (b^x)^y = b^{xy}, \tag{5}$$

где x и y — целые числа.

Если u — положительное действительное число, а m — положительное целое, то всегда существует единственное положительное действительное число v, такое, что $v^m = u$; оно называется *корнем m-й степени* из u и обозначается $v = \sqrt[m]{u}$.

Теперь определим операцию возведения в степень b^r для рациональных чисел $r = p/q$ следующим образом:

$$b^{p/q} = \sqrt[q]{b^p}. \tag{6}$$

Это определение, введенное Оремом (Oresme) (ок. 1360 г.), очень удачное, так как $b^{ap/aq} = b^{p/q}$ и правила возведения в степень остаются в силе, даже если x и y — произвольные рациональные числа (см. упр. 9).

И, наконец, определим операцию возведения в степень b^x для всех действительных чисел x. Предположим, что $b > 1$; если x определяется формулой (1), то получаем

$$b^{n+d_1/10+\cdots+d_k/10^k} \le b^x < b^{n+d_1/10+\cdots+d_k/10^k+1/10^k}. \tag{7}$$

Итак, мы *определили* положительное действительное число b^x единственным образом, так как разность между правой и левой частями соотношения (7) равна

* Ее называют мнимой единицей. — *Прим. перев.*

$b^{n+d_1/10+\cdots+d_k/10^k}(b^{1/10^k}-1)$. Как следует из упр. 13, приведенного ниже, эта разность меньше, чем $b^{n+1}(b-1)/10^k$, и если взять достаточно большое k, то можно получить значение b^x с любой заданной точностью.

Например,

$$10^{0.30102999}=1.9999999739\ldots,\quad 10^{0.30103000}=2.0000000199\,.$$

поэтому для $b=10$ и $x=0.30102999\ldots$ мы знаем значение 10^x с большей точностью, чем до одной десятимиллионной (но при этом даже не знаем, каким будет десятичное представление 10^x — $1.999\ldots$ или $2.000\ldots$).

Для $b<1$ определим операцию возведения в степень следующим образом: $b^x=(1/b)^{-x}$; если же $b=1$, то $b^x=1$. При этих определениях можно доказать, что правила возведения в степень, определяемые соотношениями (5), выполняются для всех *действительных* чисел x и y. Эти идеи определения b^x впервые были высказаны Джоном Валлисом (John Wallis) в 1655 году и Исааком Ньютоном в 1669 году.

А теперь мы подошли к одному важному вопросу. Предположим, дано положительное действительное число y. Можно ли найти такое действительное число x, что $y=b^x$? Ответ на этот вопрос утвердительный (при условии, что $b\neq 1$), так как можно просто использовать неравенства (7) *в обратном направлении*, чтобы определить n и $d_1,d_2,\ldots$ для заданного соотношения $b^x=y$. Полученное в результате число x называется *логарифмом* y *по основанию* b и записывается как $x=\log_b y$. Согласно этому определению имеем

$$x=b^{\log_b x}=\log_b(b^x). \tag{9}$$

Например, из соотношений (8) вытекает, что

$$\log_{10}2=0.30102999\ldots. \tag{10}$$

Из правил возведения в степень следует, что

$$\log_b(xy)=\log_b x+\log_b y,\quad \text{если}\quad x>0,\ y>0, \tag{11}$$

и

$$\log_b(c^y)=y\,\log_b c,\quad \text{если}\quad c>0. \tag{12}$$

В равенстве (10) приведен пример так называемого *десятичного логарифма*, т. е. логарифма по основанию 10. Можно было бы ожидать, что в работе с компьютером более удобными окажутся *двоичные логарифмы* (по основанию 2), так как основу вычислений в большинстве компьютеров составляют операции с двоичными числами. Как мы вскоре увидим, двоичные логарифмы действительно очень полезны и широко используются, но дело не только в этом. Главная причина заключается в том, что ход выполнения компьютерного алгоритма часто разветвляется на два потока. Мы будем достаточно часто использовать двоичные логарифмы, поэтому имеет смысл ввести для них краткое обозначение. Итак, следуя предложению Эдварда М. Рейнгольда (Edward M. Reingold), будем использовать запись

$$\lg x=\log_2 x. \tag{13}$$

А теперь возникает вопрос, существует ли некая связь между $\lg x$ и $\log_{10} x$. К счастью, она действительно существует. Из (9) и (12) следует, что

$$\log_{10} x = \log_{10}(2^{\lg x}) = (\lg x)(\log_{10} 2).$$

Таким образом, $\lg x = \log_{10} x / \log_{10} 2$. В общем виде эта формула выглядит следующим образом:

$$\log_c x = \frac{\log_b x}{\log_b c}. \tag{14}$$

Соотношения (11), (12) и (14) — это фундаментальные правила выполнения операций с логарифмами.

Оказывается, в большинстве случаев ни 10, ни 2 не являются идеальным основанием логарифма. Но если взять в качестве основания действительное число $e = 2.718281828459045\ldots$, то логарифмы приобретают более простые свойства. Логарифмы с основанием e принято называть *натуральными логарифмами*; при этом используется следующая запись:

$$\ln x = \log_e x. \tag{15}$$

Это нестрогое определение (в сущности, мы даже не определили число e) вряд ли позволит читателю почувствовать особую "натуральность" (т. е. естественность) такого логарифма; но по мере работы с натуральными логарифмами $\ln x$ они будут казаться нам все более естественными. Фактически натуральные логарифмы были введены Джоном Непером (John Napier) (в несколько видоизмененном виде и вне связи со степенями) еще до 1590 года, за много лет до того, как стали известны другие виды логарифмов. Следующие два примера, рассматриваемые в любом учебнике по теории вычислений, проливают свет на то, почему логарифмы Непера заслужили название "натуральных". (a) На рис. 6 площадь заштрихованной области равна $\ln x$.

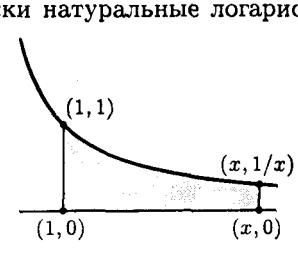


Рис. 6. Натуральный логарифм.

(b) Если банк выплачивает сложные проценты по ставке r, начисляемые каждые полгода, то прибыль на каждый доллар составит $(1+r/2)^2$ долларов; если начисление происходит каждый квартал, то вы получите $(1+r/4)^4$ долларов; если же начисление происходит каждый день, то вы получите $(1+r/365)^{365}$ долларов. Если бы проценты начислялись *непрерывно*, то вы получили бы в точности e^r долларов на каждый доллар (если не учитывать ошибку округления). В нашу компьютерную эпоху многие банки уже фактически достигли в своей работе этой предельной формулы.

История возникновения и развития понятий "логарифм" и "степень" приводится в ряде интересных статей Ф. Кэджори (F. Cajori), *AMM* **20** (1913), 5–14, 35–47, 75–84, 107–117, 148–151, 173–182, 205–210.

В заключение этого раздела выясним, как *вычислять* логарифмы. Один способ непосредственно вытекает из соотношения (7): если положить $b^x = y$ и возвести все части этого соотношения в степень 10^k, то для некоторого целого m получим

$$b^m \le y^{10^k} < b^{m+1}. \tag{16}$$

Таким образом, все, что нам нужно сделать для получения логарифма y, — возвести y в эту огромную степень и найти такое m, чтобы результат лежал между m- и $m+1$-й степенями b. Тогда искомый результат будет равен $m/10^k$ с точностью до k-го десятичного знака.

Если несколько модифицировать этот явно непрактичный метод, получим простую и удобную процедуру. Сейчас мы покажем, как вычислить $\log_{10} x$ и выразить результат в *двоичной* системе:

$$\log_{10} x = n + b_1/2 + b_2/4 + b_3/8 + \cdots. \tag{17}$$

Сначала сдвинем десятичную точку в числе x влево или вправо, чтобы получить $1 \le x/10^n < 10$; таким образом мы определим целую часть числа $\log_{10} x$, т. е. n. Чтобы найти значения b_1, b_2, ..., положим $x_0 = x/10^n$ и для $k \ge 1$ получим

$$\begin{aligned} &b_k = 0, \quad x_k = x_{k-1}^2, && \text{если} \quad x_{k-1}^2 < 10; \\ &b_k = 1, \quad x_k = x_{k-1}^2/10, && \text{если} \quad x_{k-1}^2 \ge 10. \end{aligned} \tag{18}$$

Корректность этой процедуры следует из того, что

$$1 \le x_k = x^{2^k}/10^{2^k(n+b_1/2+\cdots+b_k/2^k)} < 10 \tag{19}$$

для $k = 0, 1, 2, \ldots$, а это можно легко доказать по индукции.

На практике мы должны проводить вычисления только с конечной точностью, поэтому нельзя считать равенство $x_k = x_{k-1}^2$ точным. На самом деле можно считать, что $x_k = x_{k-1}^2$ только *приближенно* с точностью до некоторого десятичного знака. Например, приведем расчеты для $\log_{10} 2$, выполненные с точностью до четырех значащих цифр:

$$\begin{array}{llllllll} x_0 = 2.000; & & & & & & & \\ x_1 = 4.000, & b_1 = 0; & & x_6 = 1.845, & b_6 = 1; & & & \\ x_2 = 1.600, & b_2 = 1; & & x_7 = 3.404, & b_7 = 0; & & & \\ x_3 = 2.560, & b_3 = 0; & & x_8 = 1.159, & b_8 = 1; & & & \\ x_4 = 6.554, & b_4 = 0; & & x_9 = 1.343, & b_9 = 0; & & & \\ x_5 = 4.295, & b_5 = 1; & & x_{10} = 1.804, & b_{10} = 0 & \text{и т. д.} & & \end{array}$$

Погрешность вычислений приводит к росту и распространению ошибок; так, например, истинное округленное значение x_{10} равно 1.798. Накопление ошибок в конечном счете приведет к тому, что b_{19} будет вычислено неточно, и мы получим двоичное представление $(0.0100110100010000011\ldots)_2$, соответствующее десятичному представлению $0.301031\ldots$, а не истинному значению, которое дается в равенстве (10).

Для каждого подобного метода необходимо оценивать величину погрешности вычислений, возникающей из-за ошибок округления. В упр. 27 устанавливается верхняя граница погрешности; а если проводить вычисления с точностью до четырех чисел после десятичной точки, как было показано в примере выше, то получим гарантию, что значение логарифма будет вычислено с погрешностью, не превышающей 0.00044. Первоначально полученное значение логарифма является более точным потому, что значения x_0, x_1, x_2 и x_3 были вычислены *точно*.

Приведенный метод прост и довольно интересен, но, скорее всего, это не самый лучший способ вычисления логарифмов на компьютере. Еще один метод описан в упр. 25.

УПРАЖНЕНИЯ

1. [*00*] Чему равно наименьшее положительное рациональное число?

2. [*00*] Может ли выражение $1 + 0.239999999\ldots$ быть десятичным представлением действительного числа?

3. [*02*] Чему равно $(-3)^{-3}$?

▶ **4.** [*05*] Чему равно $(0.125)^{-2/3}$?

5. [*05*] Мы определили действительные числа в терминах десятичного представления. Подумайте, как можно определить их в терминах двоичного представления, и приведите аналог соотношения (2).

6. [*10*] Пусть $x = m + 0.d_1d_2\ldots$ и $y = n + 0.e_1e_2\ldots$ — действительные числа. Сформулируйте правило, которое на основе десятичного представления позволяет определить, какое из неравенств верно: $x = y$, $x < y$ или $x > y$.

7. [*M23*] Для заданных целых чисел x и y докажите правила возведения в степень на основе определения (4).

8. [*25*] Пусть m — целое положительное число. *Докажите*, что у любого положительного действительного числа u есть единственный положительный корень m-й степени, описав метод последовательного вычисления элементов десятичного представления этого корня: n, d_1, d_2, ...

9. [*M23*] Для заданных рациональных чисел x и y докажите правила возведения в степень, предполагая, что эти правила выполняются для целых чисел x и y.

10. [*18*] Докажите, что $\log_{10} 2$ не является рациональным числом.

▶ **11.** [*10*] Если $b = 10$ и $x \approx \log_{10} 2$, то сколько десятичных знаков числа x нужно знать, чтобы определить три первых десятичных знака в десятичном представлении b^x? [*Замечание.* Можете использовать результаты упр. 10.]

12. [*02*] Объясните, почему равенство (10) следует из равенств (8).

▶ **13.** [*M23*] (a) Пусть x — положительное действительное число, а n — положительное целое число. Докажите неравенство $\sqrt[n]{1+x} - 1 \le x/n$. (b) Используйте полученный результат для доказательства утверждения, следующего за соотношением (7).

14. [*15*] Докажите равенство (12).

15. [*10*] Докажите или опровергните следующее равенство:

$$\log_b x/y = \log_b x - \log_b y \quad \text{при} \quad x, y > 0.$$

16. [*00*] Как можно выразить $\log_{10} x$ через $\ln x$ и $\ln 10$?

▶ **17.** [*05*] Чему равны $\lg 32$, $\log_\pi \pi$, $\ln e$, $\log_b 1$ и $\log_b(-1)$?

18. [*10*] Докажите или опровергните следующее равенство: $\log_8 x = \frac{1}{2}\lg x$.

▶ **19.** [*20*] Поместится ли целое число n, десятичное представление которого состоит из 14 цифр, в компьютерном слове емкостью 47 бит плюс бит знака?

20. [*10*] Существует ли простое соотношение, связывающее $\log_{10} 2$ и $\log_2 10$?

21. [*15*] (*Логарифмы от логарифмов.*) Выразите $\log_b \log_b x$ через $\ln\ln x$, $\ln\ln b$ и $\ln b$.

▸ **22.** [*20*] (P. В. Хемминг (R. W. Hamming).) Докажите, что

$$\lg x \approx \ln x + \log_{10} x$$

с погрешностью, не превышающей 1%! (Таким образом, таблицы натуральных и десятичных логарифмов можно использовать и для получения приближенных значений двоичных логарифмов.)

23. [*M25*] С помощью рис. 6 дайте *геометрическое* доказательство того, что

$$\ln xy = \ln x + \ln y.$$

24. [*15*] Объясните, как нужно модифицировать метод вычисления логарифмов по основанию 10, приведенный в конце раздела, чтобы его можно было применить для вычисления логарифмов по основанию 2.

25. [*22*] Предположим, что у нас есть двоичный компьютер (т. е. компьютер, в котором используется двоичная система счисления. — *Прим. перев.*) и имеется некоторое число x, $1 \le x < 2$. Покажите, что следующий алгоритм, в котором используются только операции сдвига, сложения и вычитания в объеме, пропорциональном числу разрядов, которое определяется требуемой степенью точности, можно применить для приближенного вычисления $y = \log_b x$.

L1. [Инициализация.] Присвоить $y \leftarrow 0$, $z \leftarrow x$ с помощью сдвига вправо на 1, $k \leftarrow 1$.

L2. [Проверка окончания.] Если $x = 1$, то прекратить выполнение.

L3. [Сравнение.] Если $x - z < 1$, то присвоить $z \leftarrow z$ с помощью сдвига вправо на 1, $k \leftarrow k + 1$, и повторить этот шаг.

L4. [Замещение значений.] Присвоить $x \leftarrow x - z$, $z \leftarrow x$ с помощью сдвига вправо на k, $y \leftarrow y + \log_b(2^k/(2^k - 1))$ и перейти к шагу L2. ▮

[*Замечание.* Описанный метод очень похож на тот метод, который используется в компьютерах для выполнения операции деления. Эта идея, в сущности, принадлежит Генри Бриггсу (Henry Briggs); он применял данный метод (только не к двоичным, а к десятичным операциям) для вычисления таблиц логарифмов, опубликованных в 1624 году. Нам нужна дополнительная таблица констант $\log_b 2$, $\log_b(4/3)$, $\log_b(8/7)$ и т. д. до значения, равного точности компьютера. В данном алгоритме преднамеренно делается ошибка, так как числа сдвигаются вправо с тем, чтобы в конечном счете x свелось к 1 и выполнение алгоритма прекратилось. В этом упражнении вы должны показать, что алгоритм конечен и вычисляет приближенное значение $\log_b x$.]

26. [*M27*] Определите верхние границы точности алгоритма из предыдущего упражнения, взяв за основу точность, используемую в арифметических операциях.

▸ **27.** [*M25*] Рассмотрим метод вычисления $\log_{10} x$, описанный в этом разделе. Пусть x'_k — вычисленное приближенное значение x_k, причем x_0 определяется из соотношения $x(1-\delta) \le 10^n x'_0 \le x(1+\epsilon)$, а x'_k — из соотношений (18), где выражение $(x'_{k-1})^2$ нужно заменить на y_k и $(x'_{k-1})^2(1-\delta) \le y_k \le (x'_{k-1})^2(1+\epsilon)$, где $1 \le y_k < 100$. В этих формулах δ и ϵ — малые константы, соответствующие верхней и нижней границам погрешностей округления. Покажите, что если результат вычислений обозначить $\log' x$, то через k шагов мы получим

$$\log_{10} x + 2\log_{10}(1-\delta) - 1/2^k < \log' x \le \log_{10} x + 2\log_{10}(1+\epsilon).$$

28. [*M30*] (P. Фейнман (R. Feynman).) Придумайте метод вычисления b^x, где $0 \le x < 1$, используя только операции сдвига, сложения и вычитания (аналогично алгоритму из упр. 25), и проанализируйте его точность.

29. [*HM20*] Пусть x — действительное число, большее, чем 1. (a) При каком действительном $b > 1$ величина $b \log_b x$ минимальна? (b) При каком *целом* $b > 1$ эта величина минимальна? (c) При каком целом $b > 1$ величина $(b+1)\log_b x$ минимальна?

1.2.3. Суммы и произведения

Пусть $a_1, a_2, \ldots$ — произвольная последовательность чисел. Часто возникает необходимость в изучении сумм вида $a_1 + a_2 + \cdots + a_n$. Такую сумму более компактно можно записать следующим образом:

$$\sum_{j=1}^{n} a_j \qquad \text{или} \qquad \sum_{1 \le j \le n} a_j. \tag{1}$$

Если n равно нулю, то по определению значение суммы тоже равно нулю. Наше определение суммы можно обобщить. Если $R(j)$ — это любое соотношение, зависящее от j, то запись

$$\sum_{R(j)} a_j \tag{2}$$

обозначает сумму всех a_j, где j — целое число, удовлетворяющее условию $R(j)$. Если таких целых чисел не существует, то значение суммы (2) по определению принимается равным нулю. Буква j в (1) и (2) — это так называемый *немой индекс* (или *индексная переменная*), который вводится только для удобства записи. Для обозначения индексов суммирования обычно используются буквы i, j, k, m, n, r, s, t (иногда с надстрочными индексами или штрихами). Занимающие много места обозначения сумм, как в (1) и (2), можно заменить более компактной записью $\sum_{j=1}^{n} a_j$ или $\sum_{R(j)} a_j$. Знак "$\sum$" и немые индексы для обозначения операции суммирования в конечных пределах были введены Ж. Фурье (J. Fourier) в 1820 году.

Строго говоря, обозначение $\sum_{1 \le j \le n} a_j$ является недостаточно четким, так как не совсем ясно, по какому индексу выполняется суммирование — по j или по n. В данном конкретном случае было бы неразумно считать это суммой значений, для которых $n \ge j$. Но можно привести более сложные примеры, в которых индекс суммирования определен недостаточно четко, как в случае $\sum_{j \le k} \binom{j+k}{2j-k}$. В подобных ситуациях отличить немые индексы от индексов, имеющих самостоятельное значение (т. е. таких, которые фигурируют не только в записи суммы), можно только по контексту. Запись из приведенного выше примера будет иметь смысл, только если либо j, либо k (но не оба) не является немым индексом, т. е. имеет самостоятельное значение.

В основном, мы будем использовать запись (2) только тогда, когда сумма *конечна*, т. е. когда только конечное число значений j удовлетворяет $R(j)$ и $a_j \neq 0$. Чтобы найти бесконечную сумму, например

$$\sum_{j=1}^{\infty} a_j \;=\; \sum_{j \ge 1} a_j \;=\; a_1 + a_2 + a_3 + \cdots,$$

которая содержит бесконечно много ненулевых слагаемых, необходимо применить методы вычислений. В этом случае выражению (2) мы будем придавать следующий

смысл:

$$\sum_{R(j)} a_j = \left(\lim_{n\to\infty} \sum_{\substack{R(j)\\ 0\le j<n}} a_j\right) + \left(\lim_{n\to\infty} \sum_{\substack{R(j)\\ -n\le j<0}} a_j\right)$$

(при условии, что оба предела существуют). Если же хотя бы один предел не существует, то бесконечная сумма *расходится*; это означает, что вычислить ее нельзя. В противном случае (если оба предела существуют) сумма является *сходящейся*.

Если под знаком "$\sum$" содержится несколько условий (больше одного), как в формуле (3), значит, должны выполняться *все* условия одновременно.

Очень важное значение имеют четыре простые алгебраические операции над суммами; знакомство с ними позволяет найти решение многих задач, поэтому сейчас мы займемся обсуждением этих операций.

a) *Распределительный закон* для произведений сумм:

$$\left(\sum_{R(i)} a_i\right)\left(\sum_{S(j)} b_j\right) = \sum_{R(i)}\left(\sum_{S(j)} a_i b_j\right). \tag{4}$$

Чтобы понять этот закон, рассмотрим частный случай:

$$\begin{aligned}\left(\sum_{i=1}^{2} a_i\right)\left(\sum_{j=1}^{3} b_j\right) &= (a_1+a_2)(b_1+b_2+b_3)\\ &= (a_1b_1+a_1b_2+a_1b_3)+(a_2b_1+a_2b_2+a_2b_3)\\ &= \sum_{i=1}^{2}\left(\sum_{j=1}^{3} a_i b_j\right).\end{aligned}$$

Обычно скобки в правой части равенства (4) опускают и двойную сумму, например $\sum_{R(i)}\left(\sum_{S(j)} a_{ij}\right)$, записывают в виде $\sum_{R(i)}\sum_{S(j)} a_{ij}$.

b) *Замена индекса*:

$$\sum_{R(i)} a_i = \sum_{R(j)} a_j = \sum_{R(p(j))} a_{p(j)}. \tag{5}$$

В этом равенстве представлены два вида преобразований. В первом случае просто происходит замена индекса суммирования i на j. Второй случай представляет больший интерес. Здесь $p(j)$ — это функция от j, задающая некоторую *перестановку* на области суммирования; точнее — для каждого целого i, удовлетворяющего соотношению $R(i)$, должно существовать единственное целое число j, удовлетворяющее соотношению $p(j) = i$. Данное условие всегда выполняется в следующих важных случаях: $p(j) = c + j$ и $p(j) = c - j$, где c — целое число, не зависящее от j. Эти случаи важны потому, что на практике они встречаются чаще всего, например

$$\sum_{1\le j\le n} a_j = \sum_{1\le j-1\le n} a_{j-1} = \sum_{2\le j\le n+1} a_{j-1}. \tag{6}$$

Советую читателю внимательно изучить этот пример.

Замену j на $p(j)$ можно выполнить не для всех *бесконечных* сумм. Такая замена всегда возможна, если $p(j) = c \pm j$ (как в примере, приведенном выше), но в других

ситуациях необходимо принять некоторые меры. [Например, см. T. M. Apostol, *Mathematical Analysis* (Reading, Mass.: Addison-Wesley, 1957), Chapter 12. Достаточным условием справедливости соотношения (5) для любой перестановки целых чисел $p(j)$ является сходимость $\sum_{R(j)} |a_j|$.]

c) *Изменение порядка суммирования*:

$$\sum_{R(i)} \sum_{S(j)} a_{ij} = \sum_{S(j)} \sum_{R(i)} a_{ij}. \tag{7}$$

Давайте рассмотрим очень простой частный случай этого равенства:

$$\sum_{R(i)} \sum_{j=1}^{2} a_{ij} = \sum_{R(i)} (a_{i1} + a_{i2}),$$

$$\sum_{j=1}^{2} \sum_{R(i)} a_{ij} = \sum_{R(i)} a_{i1} + \sum_{R(i)} a_{i2}.$$

Согласно (7) правые части данных соотношений равны, т. е.

$$\sum_{R(i)} (b_i + c_i) = \sum_{R(i)} b_i + \sum_{R(i)} c_i, \tag{8}$$

где $b_i = a_{i1}$, а $c_i = a_{i2}$.

Операция изменения порядка суммирования очень полезна, так как часто простое выражение для суммы $\sum_{R(i)} a_{ij}$ известно, а для суммы $\sum_{S(j)} a_{ij}$ — нет. Необходимость в изменении порядка суммирования возникает также в более общем случае, когда соотношение $S(j)$ зависит и от i, и от j. В подобной ситуации его можно обозначить через $S(i,j)$. Изменение порядка суммирования всегда можно выполнить, по крайней мере теоретически, следующим образом:

$$\sum_{R(i)} \sum_{S(i,j)} a_{ij} = \sum_{S'(j)} \sum_{R'(i,j)} a_{ij}, \tag{9}$$

где $S'(j)$ означает, что "существует целое i, такое, что справедливы как $R(i)$, так и $S(i,j)$"; а $R'(i,j)$ означает, что "верны как $R(i)$, так и $S(i,j)$". Например, если вычисляется сумма $\sum_{i=1}^{n} \sum_{j=1}^{i} a_{ij}$, то условие $S'(j)$ звучит так: "существует целое i, такое, что $1 \le i \le n$ и $1 \le j \le i$", т. е. $1 \le j \le n$, а $R'(i,j)$ превращается в условие $1 \le i \le n$ и $1 \le j \le i$, т. е. $j \le i \le n$. В результате получаем

$$\sum_{i=1}^{n} \sum_{j=1}^{i} a_{ij} = \sum_{j=1}^{n} \sum_{i=j}^{n} a_{ij} \tag{10}$$

[*Замечание.* Как и в случае (b) (замена индекса), операция изменения порядка суммирования *не всегда справедлива для бесконечных рядов*. Если ряд *абсолютно сходится*, т. е. если сходится $\sum_{R(i)} \sum_{S(j)} |a_{ij}|$, то можно показать, что равенства (7) и (9) верны. Кроме того, если *одно* из соотношений $R(i)$ и $S(j)$ определяет *конечную* сумму в (7) и каждая бесконечная сумма сходится, то операция изменения порядка суммирования также законна. В частности, для сходящихся бесконечных сумм соотношение (8) всегда справедливо.]

d) *Манипуляции областью суммирования.* Если $R(j)$ и $S(j)$ — произвольные соотношения, то имеем

$$\sum_{R(j)} a_j + \sum_{S(j)} a_j = \sum_{R(j) \text{ или } S(j)} a_j + \sum_{R(j) \text{ и } S(j)} a_j. \tag{11}$$

Например, если $1 \le m \le n$, то

$$\sum_{1 \le j \le m} a_j + \sum_{m \le j \le n} a_j = \left(\sum_{1 \le j \le n} a_j \right) + a_m. \tag{12}$$

Здесь область "$R(j)$ и $S(j)$" просто принимает вид "$j = m$", поэтому вторая сумма свелась только к одному слагаемому "a_m". В большинстве случаев равенства (11) или соотношения $R(j)$ и $S(j)$ одновременно выполняются только для одного-двух значений j либо вообще не существует таких j, для которых справедливы и $R(j)$, и $S(j)$. В последнем случае вторая сумма в правой части равенства (11) просто исчезает.

А теперь, когда мы изучили четыре основных правила выполнения операций над суммами, давайте рассмотрим примеры их использования.

Пример 1.

$$\begin{aligned}
\sum_{0 \le j \le n} a_j &= \sum_{\substack{0 \le j \le n \\ j \text{ четное}}} a_j + \sum_{\substack{0 \le j \le n \\ j \text{ нечетное}}} a_j && \text{согласно правилу (d)} \\
&= \sum_{\substack{0 \le 2j \le n \\ 2j \text{ четное}}} a_{2j} + \sum_{\substack{0 \le 2j+1 \le n \\ 2j+1 \text{ нечетное}}} a_{2j+1} && \text{согласно правилу (b)} \\
&= \sum_{0 \le j \le n/2} a_{2j} + \sum_{0 \le j < n/2} a_{2j+1}.
\end{aligned}$$

На последнем шаге выполняется упрощение соотношений, находящихся под знаками суммы.

Пример 2. Пусть

$$\begin{aligned}
S_1 = \sum_{i=0}^{n} \sum_{j=0}^{i} a_i a_j &= \sum_{j=0}^{n} \sum_{i=j}^{n} a_i a_j && \text{согласно правилу (c) [см. (10)]} \\
&= \sum_{i=0}^{n} \sum_{j=i}^{n} a_i a_j && \text{согласно правилу (b).}
\end{aligned}$$

Здесь выполнена замена i на j и использован тот факт, что $a_j a_i = a_i a_j$. Обозначая последнюю сумму через S_2, имеем

$$2S_1 = S_1 + S_2 = \sum_{i=0}^{n}\left(\sum_{j=0}^{i} a_i a_j + \sum_{j=i}^{n} a_i a_j\right) \qquad \text{согласно (8)}$$

$$= \sum_{i=0}^{n}\left(\left(\sum_{j=0}^{n} a_i a_j\right) + a_i a_i\right) \qquad \text{согласно правилу (d) [см. (12)]}$$

$$= \sum_{i=0}^{n}\sum_{j=0}^{n} a_i a_j + \sum_{i=0}^{n} a_i a_i \qquad \text{согласно (8)}$$

$$= \left(\sum_{i=0}^{n} a_i\right)\left(\sum_{j=0}^{n} a_j\right) + \left(\sum_{i=0}^{n} a_i^2\right) \qquad \text{согласно правилу (a)}$$

$$= \left(\sum_{i=0}^{n} a_i\right)^2 + \left(\sum_{i=0}^{n} a_i^2\right) \qquad \text{согласно правилу (b).}$$

Таким образом, мы получили важное тождество:

$$\sum_{i=0}^{n}\sum_{j=0}^{i} a_i a_j = \frac{1}{2}\left(\left(\sum_{i=0}^{n} a_i\right)^2 + \left(\sum_{i=0}^{n} a_i^2\right)\right) \tag{13}$$

Пример 3 (*Сумма геометрической прогрессии*). Предположим, что $x \neq 1$, $n \geq 0$. Тогда

$$a + ax + \cdots + ax^n = \sum_{0 \leq j \leq n} ax^j \qquad \text{по определению (2)}$$

$$= a + \sum_{1 \leq j \leq n} ax^j \qquad \text{согласно правилу (d)}$$

$$= a + x\sum_{1 \leq j \leq n} ax^{j-1} \qquad \text{согласно частному случаю правила (a)}$$

$$= a + x\sum_{0 \leq j \leq n-1} ax^j \qquad \text{согласно правилу (b) [см. (6)]}$$

$$= a + x\sum_{0 \leq j \leq n} ax^j - ax^{n+1} \qquad \text{согласно правилу (d).}$$

Сравнивая первое и последнее выражения, получаем

$$(1 - x)\sum_{0 \leq j \leq n} ax^j = a - ax^{n+1};$$

отсюда следует важная формула

$$\sum_{0 \leq j \leq n} ax^j = a\left(\frac{1 - x^{n+1}}{1 - x}\right) \tag{14}$$

Пример 4 (*Сумма арифметической прогрессии*). Предположим, что $n \geq 0$. Тогда

$$\begin{aligned}
&a + (a+b) + \cdots + (a+nb) \\
&\quad = \sum_{0\le j\le n} (a+bj) && \text{по определению (2)} \\
&\quad = \sum_{0\le n-j\le n} \big(a+b(n-j)\big) && \text{по правилу (b)} \\
&\quad = \sum_{0\le j\le n} (a+bn-bj) && \text{в результате упрощения} \\
&\quad = \sum_{0\le j\le n} (2a+bn) - \sum_{0\le j\le n} (a+bj) && \text{согласно (8)} \\
&\quad = (n+1)(2a+bn) - \sum_{0\le j\le n} (a+bj),
\end{aligned}$$

так как первая сумма — это просто $(n+1)$ одинаковых слагаемых, не зависящих от j. Теперь, приравнивая первое и последнее выражения и деля их на 2, получаем

$$\sum_{0\le j\le n} (a+bj) = a(n+1) + \tfrac{1}{2}bn(n+1). \tag{15}$$

А это равно $n+1$, умноженному на $\frac{1}{2}\big(a+(a+bn)\big)$, что можно интерпретировать как количество слагаемых, умноженное на среднее первого и последнего слагаемых.

Итак, выполняя только простые операции над суммами, мы получили важные соотношения (13)–(15). В большинстве учебников эти формулы просто *приводятся*, а затем доказываются по *индукции*. Индукция — это, конечно, идеальный способ доказательства, но он не дает никакого объяснения по поводу того, каким же образом кому-то удалось придумать саму формулу, если исключить возможность некоего внезапного озарения. Занимаясь анализом алгоритмов, мы будем сталкиваться с сотнями сумм, которые не соответствуют ни одному из известных образцов. Но, выполняя над этими суммами приведенные выше операции, мы во многих случаях сможем найти решение и без догадок.

Многие операции над суммами и другие формулы можно значительно упростить, если использовать следующее обозначение:

$$[\text{утверждение}] = \begin{cases} 1, & \text{если утверждение истинно;} \\ 0, & \text{если утверждение ложно.} \end{cases} \tag{16}$$

Например, можно записать

$$\sum_{R(j)} a_j = \sum_j a_j\,[R(j)] \tag{17}$$

Обратите внимание, что в правой части суммирование производится по *всем* целым j, и это ничего не меняет, поскольку те слагаемые, для которых утверждение $R(j)$ ложно, просто равны нулю. (Мы предполагаем, что a_j определены для всех j.)

Пользуясь введенным обозначением, можно оригинальным способом вывести правило (b) из правил (a) и (c):

$$\begin{aligned}\sum_{R(p(j))} a_{p[j]} &= \sum_j a_{p(j)}\,[R(p(j))] \\ &= \sum_j \sum_i a_i\,[R(i)]\,[i = p(j)] \\ &= \sum_i a_i\,[R(i)] \sum_j [i = p(j)]\,. \end{aligned} \tag{18}$$

Сумма по j равна 1, когда $R(i)$ истинно, если предположить, что p — это перестановка в области суммирования, как требуется в (5); таким образом, остается сумма $\sum_i a_i[R(i)]$, которую можно записать как $\sum_{R(i)} a_i$. Соотношение (5) доказано. Если же p *не* является перестановкой в области суммирования, то значение суммы $\sum_{R(p(j))} a_{p(j)}$ выражается формулой (18).

Самым известным частным случаем обозначения (16) является так называемый символ *Кронекера*,

$$\delta_{ij} \;=\; [i=j] \;=\; \begin{cases} 1, & \text{если } i = j, \\ 0, & \text{если } i \neq j, \end{cases} \tag{19}$$

введенный Леопольдом Кронекером (Leopold Kronecker) в 1868 году. Более общие обозначения типа (16) были введены К. Ю. Айверсоном (K. E. Iverson) в 1962 году, поэтому (16) часто называют *обозначением Айверсона*. [См. D. E. Knuth, *AMM* **99** (1992), 403–422.]

Для произведений величин, как и для сумм, тоже существует краткая запись:

$$\prod_{R(j)} a_j. \tag{20}$$

Так обозначается произведение всех a_j, для которых целое число j удовлетворяет соотношению $R(j)$. Если ни одного такого целого j не существует, то по определению произведение равно 1 (*а не* 0).

Правила (b), (c) и (d) справедливы для произведений $\prod$ так же, как и для сумм $\sum$, если внести в них необходимые коррективы. Несколько примеров, в которых необходимо выполнить операции над произведениями, приводится в упражнениях (они даны в конце раздела).

И в завершение этого раздела приведем еще одно обозначение кратного суммирования, которое часто бывает очень удобным. Для одного или более соотношений, зависящих от *нескольких* индексов суммирования, можно использовать один знак суммы $\sum$; это означает, что сумма берется по всем комбинациям индексов, удовлетворяющим заданным условиям, например

$$\sum_{0\le i\le n} \sum_{0\le j\le n} a_{ij} = \sum_{0\le i,j\le n} a_{ij}; \qquad \sum_{0\le i\le n} \sum_{0\le j\le i} a_{ij} = \sum_{0\le j\le i\le n} a_{ij}.$$

В этой записи ни одному индексу не отдается предпочтение над другим. Выведем с ее помощью соотношение (10) новым способом

$$\sum_{i=1}^{n}\sum_{j=1}^{i} a_{ij} = \sum_{i,j} a_{ij}[1\le i\le n][1\le j\le i] = \sum_{i,j} a_{ij}[1\le j\le n][j\le i\le n]$$
$$= \sum_{j=1}^{n}\sum_{i=j}^{n} a_{ij}\,,$$

воспользовавшись тем, что $[1\le i\le n][1\le j\le i]=[1\le j\le i\le n]=[1\le j\le n][j\le i\le n]$. Аналогично более общая форма соотношения (9) следует из тождества

$$\big[R(i)\big]\big[S(i,j)\big] \;=\; \big[R(i) \text{ и } S(i,j)\big] \;=\; \big[S'(j)\big]\big[R'(i,j)\big] \tag{21}$$

Приведем еще один пример, который демонстрирует удобство суммирования с несколькими индексами:

$$\sum_{\substack{j_1+\cdots+j_n=n\\ j_1\ge\cdots\ge j_n\ge 0}} a_{j_1\ldots j_n}. \tag{22}$$

Здесь переменная a имеет n подстрочных индексов, например для $n=5$ это выражение примет следующий вид:

$$a_{11111}+a_{21110}+a_{22100}+a_{31100}+a_{32000}+a_{41000}+a_{50000}.$$

(См. замечания о разбиении числа в разделе 1.2.1.)

УПРАЖНЕНИЯ (часть 1)

1. [*01*] Что означает запись $\sum_{1\le j\le n} a_j$ для $n=3.14$?

2. [*10*] Не пользуясь знаком суммы $\sum$, запишите эквивалент выражения

$$\sum_{0\le n\le 5}\frac{1}{2n+1},$$

а также выражения

$$\sum_{0\le n^2\le 5}\frac{1}{2n^2+1}.$$

▸ **3.** [*13*] Объясните, почему несмотря на правило (b) результаты предыдущего упражнения различны.

4. [*10*] Не пользуясь знаком "$\sum$", запишите эквиваленты каждой части равенства (10) как суммы сумм для случая $n=3$.

▸ **5.** [*HM20*] Докажите, что правило (a) справедливо для произвольного бесконечного ряда при условии, что этот ряд сходится.

6. [*HM20*] Докажите, что правило (d) справедливо для произвольного бесконечного ряда при условии, что сходятся любые три суммы из четырех.

7. [*HM23*] Покажите, что если c — любое целое число, то $\sum_{R(j)} a_j = \sum_{R(c-j)} a_{c-j}$, даже если оба ряда бесконечны.

8. [*HM25*] Приведите пример бесконечного ряда, для которого равенство (7) ложно.

▸ **9.** [*05*] Справедливо ли доказательство формулы (14) для $n=-1$?

10. [*05*] Справедливо ли доказательство формулы (14) для $n=-2$?

11. [*03*] Чему равна правая часть формулы (14) при $x = 1$?

12. [*10*] Чему равна сумма $1 + \frac{1}{7} + \frac{1}{49} + \frac{1}{343} + \cdots + \left(\frac{1}{7}\right)^n$?

13. [*10*] Используя формулу (15) и предполагая, что $m \le n$, вычислите сумму $\sum_{j=m}^{n} j$.

14. [*11*] Используя результат предыдущего упражнения, вычислите сумму $\sum_{j=m}^{n}\sum_{k=r}^{s} jk$.

▸ **15.** [*M22*] Вычислите сумму $1\times2+2\times2^2+3\times2^3+\cdots+n2^n$ для малых значений n. Заметили ли вы какую-либо закономерность в этих числах? Если нет, постарайтесь обнаружить ее с помощью действий, аналогичных тем, которые применялись при выводе формулы (14).

16. [*M22*] Не применяя метод математической индукции, докажите, что

$$\sum_{j=0}^{n} jx^j = \frac{nx^{n+2} - (n+1)x^{n+1} + x}{(x-1)^2},$$

если $x \ne 1$.

▸ **17.** [*M00*] Пусть S — множество целых чисел. Чему равна сумма $\sum_{j\in S} 1$?

18. [*M20*] Покажите, как изменить порядок суммирования в равенстве (9), если $R(i)$ — это соотношение "n кратно i", а $S(i,j)$ — это соотношение "$1 \le j < i$".

19. [*20*] Чему равна сумма $\sum_{j=m}^{n}(a_j - a_{j-1})$?

▸ **20.** [*25*] Д-р Матрица* обнаружил удивительную закономерность:

$$9\times1+2=11,\ 9\times12+3=111,\ 9\times123+4=1111,\ 9\times1234+5=11111.$$

a) Запишите это великое открытие доктора с помощью знака суммы "$\sum$".

b) В вашем ответе к п. (a), без сомнения, фигурирует число 10 как основание десятичной системы. Обобщите полученную формулу так, чтобы ее можно было применять для любого основания b.

c) Докажите формулу из п. (b) с помощью формул, выведенных в тексте раздела или в упр. 16.

▸ **21.** [*M25*] Выведите правило (d) из правил (a) и (c) с помощью обозначения Айверсона (16).

▸ **22.** [*20*] Сформулируйте аналоги равенств (5), (7), (8) и (11) для *произведений*.

23. [*10*] Объясните, почему целесообразно определить $\sum_{R(j)} a_j$ и $\prod_{R(j)} a_j$ как нуль и единицу соответственно, если соотношению $R(j)$ не удовлетворяет ни одно целое число.

24. [*20*] Предположим, что $R(j)$ верно только для конечного числа j. Индукцией по числу целых чисел, удовлетворяющих $R(j)$, докажите равенство $\log_b \prod_{R(j)} a_j = \sum_{R(j)} (\log_b a_j)$ при условии, что все $a_j > 0$.

▸ **25.** [*15*] Есть ли ошибка в следующей цепочке преобразований?

$$\left(\sum_{i=1}^{n} a_i\right)\left(\sum_{j=1}^{n} \frac{1}{a_j}\right) = \sum_{1\le i\le n}\sum_{1\le j\le n} \frac{a_i}{a_j} = \sum_{1\le i\le n}\sum_{1\le i\le n} \frac{a_i}{a_i} = \sum_{i=1}^{n} 1 = n$$

26. [*25*] Покажите, что с помощью соотношений из упр. 22 $\prod_{i=0}^{n}\prod_{j=0}^{i} a_i a_j$ можно выразить через $\prod_{i=0}^{n} a_i$.

---

* В оригинале — I. J. Matrix. — *Прим. перев.*

27. [*M20*] Обобщите результат упр. 1.2.1–9, доказав неравенство

$$\prod_{j=1}^{n}(1-a_j) \geq 1-\sum_{j=1}^{n} a_j$$

при условии, что $0 < a_j < 1$.

28. [*M22*] Найдите простую формулу для $\prod_{j=2}^{n}(1-1/j^2)$.

▶ **29.** [*M30*] (a) Выразите сумму $\sum_{i=0}^{n}\sum_{j=0}^{i}\sum_{k=0}^{j} a_i a_j a_k$, используя способ записи с несколькими индексами, который приведен в конце данного раздела. (b) Выразите эту же сумму через $\sum_{i=0}^{n} a_i$, $\sum_{i=0}^{n} a_i^2$ и $\sum_{i=0}^{n} a_i^3$ [см. формулу (13)].

▶ **30.** [*M23*] (Ж. Бине (J. Binet), 1812.) Не пользуясь индукцией, докажите тождество

$$\left(\sum_{j=1}^{n} a_j x_j\right)\left(\sum_{j=1}^{n} b_j y_j\right) = \left(\sum_{j=1}^{n} a_j y_j\right)\left(\sum_{j=1}^{n} b_j x_j\right) + \sum_{1\leq j<k\leq n}(a_j b_k - a_k b_j)(x_j y_k - x_k y_j).$$

[У этого тождества есть важный частный случай: если положить $a_j = w_j$, $b_j = \overline{z}_j$, $x_j = \overline{w}_j$, $y_j = z_j$, то для произвольных комплексных чисел $w_1, \ldots, w_n, z_1, \ldots, z_n$ выполняется равенство

$$\left(\sum_{j=1}^{n}|w_j|^2\right)\left(\sum_{j=1}^{n}|z_j|^2\right) = \left|\sum_{j=1}^{n} w_j z_j\right|^2 + \sum_{1\leq j<k\leq n}|w_j\overline{z}_k - w_k\overline{z}_j|^2.$$

Члены $|w_j\overline{z}_k - w_k\overline{z}_j|^2$ неотрицательны, поэтому знаменитое *неравенство Коши-Шварца*

$$\left(\sum_{j=1}^{n}|w_j|^2\right)\left(\sum_{j=1}^{n}|z_j|^2\right) \geq \left|\sum_{j=1}^{n} w_j z_j\right|^2$$

является следствием формулы Бине.]

31. [*M20*] С помощью формулы Бине выразите сумму $\sum_{1\leq j<k\leq n}(u_j - u_k)(v_j - v_k)$ через $\sum_{j=1}^{n} u_j v_j$, $\sum_{j=1}^{n} u_j$ и $\sum_{j=1}^{n} v_j$.

32. [*M20*] Докажите, что

$$\prod_{j=1}^{n}\sum_{i=1}^{m} a_{ij} = \sum_{1\leq i_1,\ldots,i_n\leq m} a_{i_1 1}\ldots a_{i_n n}.$$

▶ **33.** [*M30*] Однажды вечером д-р Матрица открыл формулы, которые можно считать еще более замечательными по сравнению с формулами, приведенными в упр. 20:

$$\frac{1}{(a-b)(a-c)} + \frac{1}{(b-a)(b-c)} + \frac{1}{(c-a)(c-b)} = 0,$$

$$\frac{a}{(a-b)(a-c)} + \frac{b}{(b-a)(b-c)} + \frac{c}{(c-a)(c-b)} = 0,$$

$$\frac{a^2}{(a-b)(a-c)} + \frac{b^2}{(b-a)(b-c)} + \frac{c^2}{(c-a)(c-b)} = 1,$$

$$\frac{a^3}{(a-b)(a-c)} + \frac{b^3}{(b-a)(b-c)} + \frac{c^3}{(c-a)(c-b)} = a+b+c.$$

Докажите, что эти формулы являются частными случаями более общего закона. Покажите, что если $x_1, x_2, \ldots, x_n$ — различные числа, то

$$\sum_{j=1}^{n}\left(x_j^r \Big/ \prod_{\substack{1\le k\le n\\ k\ne j}} (x_j - x_k)\right) = \begin{cases} 0, & \text{если } 0 \le r < n-1; \\ 1, & \text{если } r = n-1; \\ \sum_{j=1}^{n} x_j, & \text{если } r = n. \end{cases}$$

34. [*M25*] Для произвольного x и $1 \le m \le n$ докажите, что

$$\sum_{k=1}^{n} \frac{\prod_{1\le r\le n,\, r\ne m}(x+k-r)}{\prod_{1\le r\le n,\, r\ne k}(k-r)} = 1.$$

Например, если $n = 4$ и $m = 2$, то

$$\frac{x(x-2)(x-3)}{(-1)(-2)(-3)} + \frac{(x+1)(x-1)(x-2)}{(1)(-1)(-2)} + \frac{(x+2)x(x-1)}{(2)(1)(-1)} + \frac{(x+3)(x+1)x}{(3)(2)(1)} = 1.$$

35. [*HM20*] Запись $\sup_{R(j)} a_j$ применяется для обозначения точной верхней грани элементов a_j; при этом используется тот же принцип, что и в случае применения знаков "$\sum$" и "$\prod$". (Если $R(j)$ выполняется только для конечного числа j, то вместо записи $\sup_{R(j)} a_j$ часто используется запись $\max_{R(j)} a_j$.) Покажите, как нужно видоизменить правила (a), (b), (c) и (d) для выполнения операций с *этим* обозначением. В частности, рассмотрите аналог правила (a)

$$(\sup_{R(i)} a_i) + (\sup_{S(j)} b_j) = \sup_{R(i)}(\sup_{S(j)}(a_i + b_j))$$

и дайте соответствующее определение $\sup_{R(j)} a_j$ для случая, когда $R(j)$ не выполняется *ни для одного* j.

УПРАЖНЕНИЯ (часть 2)

Матрицы и определители (детерминанты). Приведенные ниже задачи предназначены для читателя, который имеет хотя бы общее представление об определителях и элементарной теории матриц. Детерминант можно вычислить, комбинируя определенным образом следующие операции: (a) вынесение общего множителя из строки или столбца; (b) прибавление кратного одной строки (или столбца) к другой строке (или столбцу); (c) разложение по элементам какой-либо строки (или столбца). Простейший и наиболее часто используемый вариант операции (c) относится к случаю, когда все элементы первой строки или первого столбца — нули, за исключением элемента в левом верхнем углу, который равен $+1$. Тогда первая строка и первый столбец просто вычеркиваются и вычисляется определитель оставшейся части матрицы, а это уже определитель меньшего порядка. В общем случае под алгебраическим дополнением элемента a_{ij} определителя порядка $n \times n$ понимают помноженный на $(-1)^{i+j}$ определитель порядка $(n-1) \times (n-1)$, который получается в результате вычеркивания той строки и того столбца, на пересечении которых находится элемент a_{ij}. Тогда определитель матрицы равен $\sum a_{ij} \cdot$ алгебраическое дополнение (a_{ij}), причем один из индексов (i либо j) фиксируется, а суммирование выполняется по другому индексу, который пробегает значения от 1 до n.

Если (b_{ij}) — это матрица, *обратная* матрице (a_{ij}), то каждый ее элемент b_{ij} равен алгебраическому дополнению a_{ji} (*а не* a_{ij}), деленному на определитель всей матрицы.

Особое значение имеют следующие типы матриц.

Матрица Вандермонда

$$a_{ij} = x_j^i$$

$$\begin{pmatrix} x_1 & x_2 & \dots & x_n \\ x_1^2 & x_2^2 & \dots & x_n^2 \\ \vdots & & & \vdots \\ x_1^n & x_2^n & \dots & x_n^n \end{pmatrix}$$

Комбинаторная матрица

$$a_{ij} = y + \delta_{ij}x$$

$$\begin{pmatrix} x+y & y & \dots & y \\ y & x+y & \dots & y \\ & & & \vdots \\ y & y & \dots & x+y \end{pmatrix}$$

Матрица Коши

$$a_{ij} = 1/(x_i + y_j)$$

$$\begin{pmatrix} 1/(x_1+y_1) & 1/(x_1+y_2) & \dots & 1/(x_1+y_n) \\ 1/(x_2+y_1) & 1/(x_2+y_2) & \dots & 1/(x_2+y_n) \\ \vdots & & & \vdots \\ 1/(x_n+y_1) & 1/(x_n+y_2) & \dots & 1/(x_n+y_n) \end{pmatrix}$$

36. [*M23*] Покажите, что определитель комбинаторной матрицы равен $x^{n-1}(x+ny)$.

▸ **37.** [*M24*] Покажите, что определитель матрицы Вандермонда равен

$$\prod_{1\le j\le n} x_j \prod_{1\le i<j\le n} (x_j - x_i).$$

▸ **38.** [*M25*] Покажите, что определитель матрицы Коши равен

$$\prod_{1\le i<j\le n} (x_j - x_i)(y_j - y_i) \Big/ \prod_{1\le i,j\le n} (x_i + y_j)$$

39. [*M23*] Покажите, что матрица, обратная комбинаторной, — это комбинаторная матрица с элементами $b_{ij} = (-y + \delta_{ij}(x+ny))/x(x+ny)$.

40. [*M24*] Покажите, что элементы матрицы, обратной матрице Вандермонда, имеют вид

$$b_{ij} = \left(\sum_{\substack{1\le k_1<\dots<k_{n-j}\le n \\ k_1,\dots,k_{n-j}\ne i}} (-1)^{j-1} x_{k_1} \dots x_{k_{n-j}} \right) \Big/ x_i \prod_{\substack{1\le k\le n \\ k\ne i}} (x_k - x_i).$$

Пусть вас не пугает сложная сумма в числителе — это просто коэффициент при x^{j-1} в многочлене $(x_1 - x)\dots(x_n - x)/(x_i - x)$.

41. [*M26*] Покажите, что элементы матрицы, обратной матрице Коши, имеют вид

$$b_{ij} = \left(\prod_{1\le k\le n} (x_j + y_k)(x_k + y_i) \right) \Big/ (x_j + y_i) \left(\prod_{\substack{1\le k\le n \\ k\ne j}} (x_j - x_k) \right) \left(\prod_{\substack{1\le k\le n \\ k\ne i}} (y_i - y_k) \right).$$

42. [*M18*] Чему равна сумма всех n^2 элементов матрицы, обратной комбинаторной?

43. [*M24*] Чему равна сумма всех n^2 элементов матрицы, обратной матрице Вандермонда? [*Указание.* Воспользуйтесь упр. 33.]

▸ **44.** [*M26*] Чему равна сумма всех n^2 элементов матрицы, обратной матрице Коши?

▸ **45.** [*M25*] *Матрица Гильберта*, которую иногда называют сегментом размера $n \times n$ *(бесконечной)* матрицы Гильберта, — это матрица, элементы которой имеют следующий вид: $a_{ij} = 1/(i+j-1)$. Покажите, что матрица Гильберта является частным случаем матрицы Коши; найдите для нее обратную матрицу; покажите, что все элементы этой обратной матрицы являются целыми числами и что сумма всех элементов обратной матрицы равна n^2. [*Замечание.* Матрицы Гильберта часто используются для тестирования различных алгоритмов, в которых выполняются операции над матрицами, так как, во-первых, они численно неустойчивы относительно преобразований и, во-вторых, для них известны обратные матрицы. Но было бы ошибкой сравнивать *известную* обратную матрицу, заданную в этом упражнении, с *вычисленной* обратной матрицей к матрице Гильберта, поскольку, прежде чем находить обратную матрицу, необходимо округлить элементы первоначальной матрицы. В результате из-за уже упомянутой неустойчивости обратная матрица к округленной матрице Гильберта будет несколько отличаться от точного варианта обратной матрицы. Элементы обратной матрицы являются целыми числами, т. е. их не надо округлять, поэтому обратная матрица определена точно и можно попытаться ее обратить. Но заметим, что обратная матрица так же неустойчива, как и первоначальная. Кроме того, элементами обратной матрицы являются достаточно большие числа.] Для решения данной задачи необходимо знание основных фактов о факториалах и биномиальных коэффициентах, о которых будет говориться в разделах 1.2.5 и 1.2.6.

▸ **46.** [*M30*] Пусть A — матрица размера $m \times n$, а B — матрица размера $n \times m$. При условии, что $1 \le j_1, j_2, \ldots, j_m \le n$, обозначим через $A_{j_1 j_2 \ldots j_m}$ матрицу размера $m \times m$, состоящую из столбцов $j_1, \ldots, j_m$ матрицы A, а через $B_{j_1 j_2 \ldots j_m}$ — матрицу размера $m \times m$, состоящую из строк $j_1, \ldots, j_m$ матрицы B. Докажите *тождество Бине-Коши*

$$\det(AB) = \sum_{1 \le j_1 < j_2 < \cdots < j_m \le n} \det(A_{j_1 j_2 \ldots j_m}) \det(B_{j_1 j_2 \ldots j_m}).$$

(Обратите внимание на частные случаи: (i) $m = n$, (ii) $m = 1$, (iii) $B = A^T$, (iv) $m > n$, (v) $m = 2$.)

47. [*M27*] (К. Краттенхалер (C. Krattenthaler).) Докажите, что

$$\det \begin{pmatrix} (x+q_2)(x+q_3) & (x+p_1)(x+q_3) & (x+p_1)(x+p_2) \\ (y+q_2)(y+q_3) & (y+p_1)(y+q_3) & (y+p_1)(y+p_2) \\ (z+q_2)(z+q_3) & (z+p_1)(z+q_3) & (z+p_1)(z+p_2) \end{pmatrix} = (x-y)(x-z)(y-z)(p_1-q_2)(p_1-q_3)(p_2-q_3),$$

и обобщите это равенство для определителя матрицы размера $n \times n$, зависящего от $3n-2$ переменных $x_1, \ldots, x_n, p_1, \ldots, p_{n-1}, q_2, \ldots, q_n$. Сравните полученную формулу с результатом упр. 38.

1.2.4. Целочисленные функции и элементарная теория чисел

Для произвольного действительного числа x введем следующие обозначения:

$\lfloor x \rfloor$ — наибольшее целое, которое меньше или равно x (*"пол" (floor)* x);

$\lceil x \rceil$ — наименьшее целое, которое больше или равно x (*"потолок" (ceiling)* x).

До 1970 года запись $[x]$ часто использовалась для обозначения одной из этих двух функций, но обычно для первой. Приведенные выше обозначения, введенные К. Ю. Айверсоном в начале 60-х годов, более удобны потому, что на практике функции $\lfloor x \rfloor$ и $\lceil x \rceil$ встречаются одинаково часто. Функцию $\lfloor x \rfloor$ иногда называют *целой частью* числа x.

Приведем формулы и примеры, которые очень легко проверить:

$$\lfloor\sqrt{2}\rfloor = 1, \quad \lceil\sqrt{2}\rceil = 2, \quad \lfloor+\frac{1}{2}\rfloor = 0, \quad \lceil-\frac{1}{2}\rceil = 0, \quad \lfloor-\frac{1}{2}\rfloor = -1 \text{ (а } \textit{не} \text{ нулю!)};$$

$\lceil x\rceil = \lfloor x\rfloor$ тогда и только тогда, когда x — целое число;

$\lceil x\rceil = \lfloor x\rfloor + 1$ тогда и только тогда, когда x не является целым числом;

$\lfloor -x\rfloor = -\lceil x\rceil; \quad x - 1 < \lfloor x\rfloor \le x \le \lceil x\rceil < x + 1.$

В упражнениях, приведенных в конце раздела, даны также другие важные формулы, которые связаны с операциями "пол" и "потолок"

Для произвольных действительных чисел x и y определим следующую бинарную операцию:

$$x \bmod y = x - y\lfloor x/y\rfloor, \quad \text{если } y \neq 0; \qquad x \bmod 0 = x.$$

Из этого определения видно, что если $y \neq 0$, то

$$0 \le \frac{x}{y} - \left\lfloor\frac{x}{y}\right\rfloor = \frac{x \bmod y}{y} < 1. \tag{2}$$

Следовательно,

a) если $y > 0$, то $0 \le x \bmod y < y$;

b) если $y < 0$, то $0 \ge x \bmod y > y$;

c) величина $x - (x \bmod y)$ является целочисленным кратным y.

Выражение $x \bmod y$ называется *остатком* от деления x на y; аналогично $\lfloor x/y\rfloor$ называется *частным*.

Если x и y являются целыми числами, то "mod" — это знакомая нам операция:

$$5 \bmod 3 = 2, \qquad 18 \bmod 3 = 0, \qquad -2 \bmod 3 = 1. \tag{3}$$

Имеем $x \bmod y = 0$ тогда и только тогда, когда x кратно y, т. е. тогда и только тогда, когда x делится на y. Запись $y\backslash x$ читается как "y делит x", т. е. y — это положительное целое число и $x \bmod y = 0$.

Операция "mod" применяется также в том случае, когда x и y принимают произвольные действительные значения. Например, для тригонометрических функций можно записать следующее:

$$\tan x = \tan(x \bmod \pi).$$

Величина $x \bmod 1$ — это *дробная часть* числа x. Из (1) получаем, что

$$x = \lfloor x\rfloor + (x \bmod 1). \tag{4}$$

В работах по теории чисел сокращение "mod" часто используется в несколько ином, хотя и близком смысле. Для описания теоретико-числового понятия *сравнимость* (конгруэнтность) будем применять следующую запись:

$$x \equiv y \text{ (по модулю } z). \tag{5}$$

Это означает, что $x \bmod z = y \bmod z$, т. е. $x - y$ — целое число, кратное z. Запись (5) читается следующим образом: "x сравнимо с y по модулю z".

А теперь перейдем к рассмотрению основных свойств сравнений, которые будут использоваться в доказательствах, основанных на фактах из теории чисел. Предполагается, что в приведенных ниже формулах все переменные являются целыми числами. Целые числа x и y называются *взаимно простыми*, если у них нет общих множителей, т. е. если их наибольший общий делитель равен 1. Для обозначения этого понятия будем использовать следующую запись: $x \perp y$. На самом деле понятие взаимной простоты целых чисел всем хорошо знакомо; когда мы говорим, что дробь "несократима", это означает, что числитель и знаменатель — взаимно простые числа.

Свойство A. Если $a \equiv b$ и $x \equiv y$, то $a \pm x \equiv b \pm y$ и $ax \equiv by$ (по модулю m),

Свойство B. Если $ax \equiv by$ и $a \equiv b$ и если $a \perp m$, то $x \equiv y$ (по модулю m).

Свойство C. $a \equiv b$ (по модулю m) тогда и только тогда, когда $an \equiv bn$ (по модулю mn) при $n \neq 0$.

Свойство D. Если $r \perp s$, то $a \equiv b$ (по модулю rs) тогда и только тогда, когда $a \equiv b$ (по модулю r) и $a \equiv b$ (по модулю s).

Свойство A говорит о том, что мы можем выполнять сложение, вычитание и умножение по модулю m точно так же, как при выполнении обычных операций сложения, вычитания и умножения. Свойство B касается операции деления и утверждает, что операция сравнения позволяет также выполнять сокращение на общий множитель в случае, если этот множитель и модуль — взаимно простые числа.

Свойства C и D указывают на то, что происходит при изменения модуля. Они доказываются в приведенных ниже упражнениях.

Следующая важная теорема является следствием свойств A и B.

Теорема F (*Теорема Ферма*, 1640). *Если p — простое число, то $a^p \equiv a$ (по модулю p) для всех целых a.*

Доказательство. Если a кратно p, то, очевидно, $a^p \equiv 0 \equiv a$ (по модулю p). Поэтому достаточно рассмотреть случай, когда $a \bmod p \neq 0$. Так как p — простое число, то $a \perp p$. Рассмотрим следующие числа:

$$0 \bmod p, \quad a \bmod p, \quad 2a \bmod p, \quad \ldots, \quad (p-1)a \bmod p. \tag{6}$$

Все эти p чисел *различны*, так как если бы $ax \bmod p = ay \bmod p$, то по определению (5) $ax \equiv ay$ (по модулю p); следовательно, согласно свойству B $x \equiv y$ (по модулю p).

Таким образом, последовательность (6) представляет собой p различных неотрицательных чисел, меньших, чем p; причем первое число — нуль, а все остальные — упорядоченные определенным образом целые числа $1, 2, \ldots, p-1$. Следовательно, согласно свойству A

$$(a)(2a) \ldots \big((p-1)a\big) \equiv 1 \cdot 2 \ldots (p-1) \text{ (по модулю } p). \tag{7}$$

Умножая обе части этого сравнения на a, получим

$$a^p\big(1 \cdot 2 \ldots (p-1)\big) \equiv a\big(1 \cdot 2 \ldots (p-1)\big) \text{ (по модулю } p). \tag{8}$$

Это доказывает теорему, поскольку каждый из множителей $1, 2, \ldots, p-1$ взаимно прост с p, поэтому на основании свойства B его можно сократить. ∎

УПРАЖНЕНИЯ

1. [*00*] Чему равны $\lfloor 1.1 \rfloor$, $\lfloor -1.1 \rfloor$, $\lceil -1.1 \rceil$, $\lfloor 0.99999 \rfloor$ и $\lfloor \lg 35 \rfloor$?

▶ **2.** [*01*] Чему равно $\lceil \lfloor x \rfloor \rceil$?

3. [*M10*] Пусть n — целое, а x — действительное число. Докажите, что:

a) $\lfloor x \rfloor < n$ тогда и только тогда, когда $x < n$;

b) $n \le \lfloor x \rfloor$ тогда и только тогда, когда $n \le x$;

c) $\lceil x \rceil \le n$ тогда и только тогда, когда $x \le n$;

d) $n < \lceil x \rceil$ тогда и только тогда, когда $n < x$;

e) $\lfloor x \rfloor = n$ тогда и только тогда, когда $x - 1 < n \le x$, а также тогда и только тогда, когда $n \le x < n + 1$;

f) $\lceil x \rceil = n$ тогда и только тогда, когда $x \le n < x + 1$, а также тогда и только тогда, когда $n - 1 < x \le n$.

[*Эти формулы являются самыми важными инструментами доказательства утверждений, касающихся* $\lfloor x \rfloor$ *и* $\lceil x \rceil$.]

▶ **4.** [*M10*] Используя предыдущее упражнение, докажите, что $\lfloor -x \rfloor = -\lceil x \rceil$.

5. [*16*] Пусть x — положительное действительное число. Найдите простую формулу *округления* x *до ближайшего целого числа*. Искомое правило округления должно давать $\lfloor x \rfloor$ в случае, если $x \bmod 1 < \frac{1}{2}$, и $\lceil x \rceil$, если $x \bmod 1 \ge \frac{1}{2}$. Вы должны получить единую формулу, включающую в себя оба случая. Рассмотрите, как ваша формула будет выполнять округление для отрицательного x.

▶ **6.** [*20*] Какие из следующих равенств верны для всех положительных действительных чисел x: (a) $\lfloor \sqrt{\lfloor x \rfloor} \rfloor = \lfloor \sqrt{x} \rfloor$; (b) $\lceil \sqrt{\lceil x \rceil} \rceil = \lceil \sqrt{x} \rceil$; (c) $\lceil \sqrt{\lfloor x \rfloor} \rceil = \lceil \sqrt{x} \rceil$?

7. [*M15*] Покажите, что $\lfloor x \rfloor + \lfloor y \rfloor \le \lfloor x + y \rfloor$ и что это соотношение становится равенством тогда и только тогда, когда $x \bmod 1 + y \bmod 1 < 1$. Выполняется ли аналогичная формула для функции "потолок"?

8. [*00*] Чему равно $100 \bmod 3$, $100 \bmod 7$, $-100 \bmod 7$, $-100 \bmod 0$?

9. [*05*] Чему равно $5 \bmod -3$, $18 \bmod -3$, $-2 \bmod -3$?

▶ **10.** [*10*] Чему равно $1.1 \bmod 1$, $0.11 \bmod .1$, $0.11 \bmod -.1$?

11. [*00*] Что означает выражение "$x \equiv y$ (по модулю 0)" согласно принятому нами определению?

12. [*00*] Какие целые числа взаимно просты с 1?

13. [*M00*] Будем считать, что наибольший общий делитель чисел 0 и n равен $|n|$. Какие целые числа взаимно просты с 0?

▶ **14.** [*12*] Если $x \bmod 3 = 2$ и $x \bmod 5 = 3$, чему равно $x \bmod 15$?

15. [*10*] Докажите, что $z(x \bmod y) = (zx) \bmod (zy)$. [Правило C немедленно следует из этого распределительного закона.]

16. [*M10*] Пусть $y > 0$. Покажите, что если $(x - z)/y$ — целое число и $0 \le z < y$, то $z = x \bmod y$.

17. [*M15*] Выведите свойство A непосредственно из определения сравнимости и докажите первую половину свойства D: если $a \equiv b$ (по модулю rs), то $a \equiv b$ (по модулю r) и $a \equiv b$ (по модулю s). (Здесь r и s — произвольные целые числа.)

18. [*M15*] С помощью свойства B докажите вторую половину свойства D: если $a \equiv b$ (по модулю r) и $a \equiv b$ (по модулю s), то $a \equiv b$ (по модулю rs) при условии, что $r \perp s$.

▶ **19.** [*M10*] (*Закон существования обратного элемента.*) Если $n \perp m$, то существует такое целое n', что $nn' \equiv 1$ (по модулю m). Докажите это с помощью обобщенного алгоритма Евклида (алгоритм 1.2.1E).

20. [*M15*] Докажите свойство B с помощью свойства A и закона существования обратного элемента.

21. [*M22*] (*Основная теорема арифметики.*) С помощью свойства B и упр. 1.2.1–5 докажите, что любое целое число $n > 1$ можно *единственным образом* (если не учитывать порядок сомножителей) представить в виде произведения простых чисел. Другими словами, покажите, что существует ровно один способ записи $n = p_1 p_2 \ldots p_k$, где каждое p_j — простое число и $p_1 \le p_2 \le \cdots \le p_k$.

▶ **22.** [*M10*] Приведите пример того, что свойство B не всегда выполняется в случае, если a и m не являются взаимно простыми.

23. [*M10*] Приведите пример того, что свойство D не всегда выполняется, если r и s не являются взаимно простыми.

▶ **24.** [*M20*] Допускают ли свойства A, B, C и D такое обобщение, чтобы они выполнялись не только для целых, но и для произвольных действительных чисел?

25. [*M02*] На основании теоремы F покажите, что $a^{p-1} \bmod p = [a \text{ не кратно } p]$ для любого простого числа p.

26. [*M15*] Пусть p — нечетное простое число, a — произвольное целое и $b = a^{(p-1)/2}$. Покажите, что $b \bmod p$ равно либо 0, либо 1, либо $p-1$. [*Указание.* Рассмотрите $(b+1)(b-1)$.]

27. [*M15*] Пусть n — положительное целое число и пусть $\varphi(n)$ — количество значений из множества $\{0, 1, \ldots, n-1\}$, взаимно простых с n. Таким образом, $\varphi(1) = 1$, $\varphi(2) = 1$, $\varphi(3) = 2$, $\varphi(4) = 2$ и т. д. Покажите, что если p — простое число, то $\varphi(p) = p - 1$, и вычислите $\varphi(p^e)$, где e — положительное целое число.

▶ **28.** [*M25*] Покажите, что метод, который использовался для доказательства теоремы F, можно применить для доказательства следующего ее обобщения, называемого *теоремой Эйлера*: если $a \perp m$, то $a^{\varphi(m)} \equiv 1$ (по модулю m) для *любого* положительного целого m. (В частности, число n' из упр. 19 можно взять равным $n^{\varphi(m)-1} \bmod m$.)

29. [*M22*] Функция $f(n)$ от положительного целочисленного аргумента n называется *мультипликативной* (*multiplicative*), если $f(rs) = f(r)f(s)$ для любых взаимно простых r и s. Покажите, что каждая из перечисленных ниже функций является мультипликативной: (a) $f(n) = n^c$, где c — произвольная постоянная; (b) $f(n) = [n$ не делится на k^2 для любого целого $k > 1]$; (c) $f(n) = c^k$, где k — количество различных простых чисел, которые делят n; (d) произведение любых двух мультипликативных функций.

30. [*M30*] Докажите, что функция $\varphi(n)$ из упр. 27 является мультипликативной. Используя этот факт, вычислите $\varphi(1000000)$ и предложите простой метод вычисления $\varphi(n)$, когда n разложено на простые множители.

31. [*M22*] Докажите, что если функция $f(n)$ мультипликативна, то $g(n) = \sum_{d \backslash n} f(d)$ также мультипликативна.

32. [*M18*] Докажите, что для любой функции $f(x, y)$ выполняется тождество

$$\sum_{d \backslash n} \sum_{c \backslash d} f(c, d) = \sum_{c \backslash n} \sum_{d \backslash (n/c)} f(c, cd).$$

33. [*M18*] Для целых чисел m и n вычислите:
(a) $\lfloor \frac{1}{2}(n+m) \rfloor + \lfloor \frac{1}{2}(n-m+1) \rfloor$; (b) $\lceil \frac{1}{2}(n+m) \rceil + \lceil \frac{1}{2}(n-m+1) \rceil$
(Обратите внимание на частный случай $m = 0$.)

▶ **34.** [*M21*] Какие необходимые и достаточные условия нужно наложить на действительное число $b > 1$, чтобы равенство $\lfloor \log_b x \rfloor = \lfloor \log_b \lfloor x \rfloor \rfloor$ выполнялось для всех действительных $x \ge 1$?

▸ **35.** [*M20*] Пусть m и n — целые числа и $n > 0$. Докажите, что равенство

$$\lfloor (x+m)/n \rfloor = \lfloor (\lfloor x \rfloor + m)/n \rfloor$$

выполняется для всех действительных x. (Заметим, что $m = 0$ — это очень важный частный случай.) Будет ли справедливо аналогичное равенство для функции "потолок"?

36. [*M23*] Докажите, что $\sum_{k=1}^{n} \lfloor k/2 \rfloor = \lfloor n^2/4 \rfloor$; вычислите также сумму $\sum_{k=1}^{n} \lceil k/2 \rceil$

▸ **37.** [*M30*] Пусть m и n — целые числа, $n > 0$. Покажите, что

$$\sum_{0 \le k < n} \left\lfloor \frac{mk+x}{n} \right\rfloor = \frac{(m-1)(n-1)}{2} + \frac{d-1}{2} + d\lfloor x/d \rfloor,$$

где d — наибольший общий делитель m и n, а x — любое действительное число.

38. [*M26*] (E. Буше (E. Busche), 1909.) Докажите, что для всех действительных чисел x и y, причем $y > 0$:

$$\sum_{0 \le k < y} \left\lfloor x + \frac{k}{y} \right\rfloor = \lfloor xy + \lfloor x+1 \rfloor (\lceil y \rceil - y) \rfloor.$$

В частности, если y — целое положительное число, то, обозначив его через n, получим важную формулу:

$$\lfloor x \rfloor + \left\lfloor x + \frac{1}{n} \right\rfloor + \cdots + \left\lfloor x + \frac{n-1}{n} \right\rfloor = \lfloor nx \rfloor.$$

39. [*HM35*] Функция f, для которой $f(x) + f(x + \frac{1}{n}) + \cdots + f(x + \frac{n-1}{n}) = f(nx)$, где n — любое положительное целое число, называется *репликативной функцией*. Из предыдущего упражнения следует, что функция $\lfloor x \rfloor$ репликативна. Покажите, что репликативны следующие функции:

a) $f(x) = x - \frac{1}{2}$;
b) $f(x) = [x$ — целое число$]$;
c) $f(x) = [x$ — положительное целое число$]$;
d) $f(x) = [$существует рациональное число r и целое m, такие, что $x = r\pi + m]$;
e) еще три функции, аналогичные функции из (d), для которых r и/или m рассматриваются только на множестве положительных чисел;
f) $f(x) = \log|2 \sin \pi x|$, если допускается значение $f(x) = -\infty$;
g) сумма любых двух репликативных функций;
h) произведение репликативной функции на константу;
i) функция $g(x) = f(x - \lfloor x \rfloor)$, где $f(x)$ репликативна.

40. [*HM46*] Исследуйте класс репликативных функций; определите все репликативные функции специального типа. Например, является ли функция из п. (a) в упр. 39 единственной непрерывной репликативной функцией? Интересно рассмотреть также более общий класс функций, для которых

$$f(x) + f\left(x + \frac{1}{n}\right) + \cdots + f\left(x + \frac{n-1}{n}\right) = a_n f(nx) + b_n$$

Здесь a_n и b_n — это числа, которые зависят от n, но не зависят от x. Производные и интегралы от данных функций (если $b_n = 0$) относятся к этому же типу. Если мы потребуем, чтобы $b_n = 0$, то получим, например, многочлены Бернулли, тригонометрические функции $\cot \pi x$ и $\csc^2 \pi x$, а также обобщенную дзета-функцию Гурвица $\zeta(s, x) = \sum_{k \ge 0} 1/(k+x)^s$, где s фиксировано. При $b_n \neq 0$ получим другие хорошо известные функции, например пси-функцию.

41. [*M23*] Пусть $a_1, a_2, a_3, \ldots$ — последовательность 1, 2, 2, 3, 3, 3, 4, 4, 4, 4, ... Выразите a_n как функцию от n с помощью функций "пол" и/или "потолок".

42. [*M24*] (a) Докажите, что

$$\sum_{k=1}^{n} a_k = na_n - \sum_{k=1}^{n-1} k(a_{k+1} - a_k), \qquad \text{если } n > 0.$$

(b) Предыдущая функция используется для вычисления сумм, в которых присутствует функция "пол". Докажите, что если b — целое число и $b \geq 2$, то

$$\sum_{k=1}^{n} \lfloor \log_b k \rfloor = (n+1)\lfloor \log_b n \rfloor - (b^{\lfloor \log_b n \rfloor + 1} - b)/(b-1).$$

43. [*M23*] Вычислите сумму $\sum_{k=1}^{n} \lfloor \sqrt{k} \rfloor$.

44. [*M24*] Покажите, что $\sum_{k\geq 0} \sum_{1\leq j<b} \lfloor (n + jb^k)/b^{k+1} \rfloor = n$, если b и n — целые числа, $n \geq 0$ и $b \geq 2$. Чему равна эта сумма при $n < 0$?

▸ **45.** [*M28*] Результат упр. 37 несколько удивляет, так как из него следует, что

$$\sum_{0\leq k<n} \left\lfloor \frac{mk+x}{n} \right\rfloor = \sum_{0\leq k<m} \left\lfloor \frac{nk+x}{m} \right\rfloor.$$

Это "соотношение взаимности" — только один пример из множества подобных формул (см. раздел 3.3.3). Покажите, что для любой функции f

$$\sum_{0\leq j<n} f\left(\left\lfloor \frac{mj}{n} \right\rfloor\right) = \sum_{0\leq r<m} \left\lceil \frac{rn}{m} \right\rceil (f(r-1) - f(r)) + nf(m-1).$$

В частности, докажите, что

$$\sum_{0\leq j<n} \binom{\lfloor mj/n \rfloor + 1}{k} + \sum_{0\leq j<m} \left\lceil \frac{jn}{m} \right\rceil \binom{j}{k-1} = n\binom{m}{k}.$$

[*Указание.* Выполните замену переменных $r = \lfloor mj/n \rfloor$. Биномиальные коэффициенты $\binom{m}{k}$ обсуждаются в разделе 1.2.6.]

46. [*M29*] (*Обобщенный закон взаимности.*) Обобщите формулу из упр. 45 таким образом, чтобы получить выражение для суммы $\sum_{0\leq j<\alpha n} f(\lfloor mj/n \rfloor)$, где α — *любое* положительное действительное число.

▸ **47.** [*M31*] Пусть p — нечетное простое число. Определим *символ Лежандра*, $\left(\frac{q}{p}\right)$, считая его равным $+1$, 0 или -1, в зависимости от того, чему равно $q^{(p-1)/2} \bmod p$: 1, 0 или $p-1$. (См. упр. 26.)

a) Пусть q не кратно p. Покажите, что числа

$$(-1)^{\lfloor 2kq/p \rfloor}(2kq \bmod p), \qquad 0 < k < p/2,$$

сравнимы по модулю с числами $2, 4, \ldots, p-1$, взятыми в определенном порядке. Следовательно, $\left(\frac{q}{p}\right) = (-1)^{\sigma}$, где $\sigma = \sum_{0\leq k<p/2} \lfloor 2kq/p \rfloor$.

b) Используйте результат (a) для вычисления $\left(\frac{2}{p}\right)$.

c) Пусть q — нечетное число. Покажите, что

$$\textstyle\sum_{0\leq k<p/2} \lfloor 2kq/p \rfloor \equiv \sum_{0\leq k<p/2} \lfloor kq/p \rfloor \text{ (по модулю 2).}$$

[*Указание.* Рассмотрите величину $\lfloor (p-1-2k)q/p \rfloor$.]

d) Воспользуйтесь общей формулой взаимности из упр. 46, чтобы получить *закон квадратичной взаимности*: $\left(\frac{q}{p}\right)\left(\frac{p}{q}\right) = (-1)^{(p-1)(q-1)/4}$, если p и q — различные нечетные простые числа.

48. [*M26*] Пусть m и n — целые числа. Докажите или опровергните следующие тождества:

$$\text{(a)}\ \left\lfloor \frac{m+n-1}{n} \right\rfloor = \left\lceil \frac{m}{n} \right\rceil; \qquad \text{(b)}\ \left\lfloor \frac{n+2-\lfloor n/25 \rfloor}{3} \right\rfloor = \left\lfloor \frac{8n+24}{25} \right\rfloor.$$

49. [*M30*] Предположим, что целочисленная функция $f(x)$ удовлетворяет следующим двум простым условиям: (i) $f(x+1) = f(x)+1$; (ii) для всех положительных целых n $f(x) = f(f(nx)/n)$. Докажите, что для всех рациональных x либо $f(x) = \lfloor x \rfloor$, либо $f(x) = \lceil x \rceil$.

1.2.5. Перестановки и факториалы

Перестановкой n объектов называется способ последовательного расположения n различных объектов с учетом порядка. Например, для трех объектов $\{a, b, c\}$ существует шесть перестановок:

$$a\,b\,c, \qquad a\,c\,b, \qquad b\,a\,c, \qquad b\,c\,a, \qquad c\,a\,b, \qquad c\,b\,a. \tag{1}$$

Свойства перестановок играют очень важную роль в анализе алгоритмов; далее в этой книге мы установим много интересных фактов, касающихся перестановок (permutation)*. А пока наша задача — просто *сосчитать* их, т. е. выяснить, сколько имеется возможных перестановок для n объектов. Существует n способов выбора крайнего объекта слева (первого); после того как этот выбор сделан, существует $n-1$ способ выбора следующего за ним объекта. Таким образом, получаем $n(n-1)$ способов выбора объектов для первых двух позиций. Аналогично третий объект можно выбрать $n-2$ способами, что в итоге дает $n(n-1)(n-2)$ возможных способов выбора первых трех объектов. В общем случае, обозначив через p_{nk} количество способов выбора k объектов из n с учетом порядка, получим

$$p_{nk} = n(n-1)\ldots(n-k+1). \tag{2}$$

Отсюда вытекает, что общее число перестановок выражается формулой

$$p_{nn} = n(n-1)\ldots(1).$$

Для наших прикладных целей очень важное значение имеет процесс *построения* всех перестановок из n объектов методом индукции в предположении, что все перестановки из $n-1$ объектов уже построены. Давайте перепишем (1), заменив буквы $\{a, b, c\}$ цифрами $\{1, 2, 3\}$; тогда получим следующие перестановки:

$$1\,2\,3, \qquad 1\,3\,2, \qquad 2\,1\,3, \qquad 2\,3\,1, \qquad 3\,1\,2, \qquad 3\,2\,1. \tag{3}$$

А теперь давайте подумаем, как из этого набора получить все возможные перестановки из четырех цифр $\{1, 2, 3, 4\}$. Существует два основных метода перехода от перестановок из $n-1$ объектов к перестановкам из n объектов.

Метод 1. Для каждой перестановки $a_1\,a_2\,\ldots\,a_{n-1}$ из $\{1, 2, \ldots, n-1\}$ объектов построим еще n перестановок, помещая число n на все возможные места, в результате чего получим

$$n\,a_1\,a_2\,\ldots\,a_{n-1}, \quad a_1\,n\,a_2\,\ldots\,a_{n-1}, \quad \ldots, \quad a_1\,a_2\,\ldots\,n\,a_{n-1}, \quad a_1\,a_2\,\ldots\,a_{n-1}\,n.$$

* В связи с чрезвычайной важностью перестановок Воган Пратт (Vaughan Pratt) предложил для краткости называть их "perms". Как только предложение Пратта будет принято, учебники по программированию немного "похудеют" (и, возможно, подешевеют).

Например, для перестановки 2 3 1 из (3) получим следующее: 4 2 3 1, 2 4 3 1, 2 3 4 1, 2 3 1 4. Очевидно, это все возможные перестановки из n объектов, причем ни одна из них не повторяется.

Метод 2. Для каждой перестановки $a_1\,a_2\,\ldots\,a_{n-1}$ из $\{1,2,\ldots,n-1\}$ объектов построим еще n перестановок следующим образом. Сначала построим набор

$$a_1\,a_2\,\ldots\,a_{n-1}\,\tfrac{1}{2},\quad a_1\,a_2\,\ldots\,a_{n-1}\,\tfrac{3}{2},\quad \ldots,\quad a_1\,a_2\,\ldots\,a_{n-1}\,\left(n-\tfrac{1}{2}\right).$$

Затем заменим элементы каждой перестановки цифрами $\{1,2,\ldots,n\}$, *сохраняя порядок*. Например, для перестановки 2 3 1 из (3) получим

$$2\,3\,1\,\tfrac{1}{2},\quad 2\,3\,1\,\tfrac{3}{2},\quad 2\,3\,1\,\tfrac{5}{2},\quad 2\,3\,1\,\tfrac{7}{2}$$

и после замены получим

$$3\,4\,2\,1,\quad 3\,4\,1\,2,\quad 2\,4\,1\,3,\quad 2\,3\,1\,4.$$

Есть еще один способ описания этого процесса. Возьмем перестановку $a_1a_2\ldots a_{n-1}$ и число k, $1\le k\le n$; добавим единицу к каждому a_j, значение которого $\ge k$. В результате получим перестановку $b_1b_2\ldots b_{n-1}$ из элементов $\{1,\ldots,k-1,k+1,\ldots,n\}$; значит, $b_1b_2\ldots b_{n-1}k$ — это перестановка чисел $\{1,\ldots,n\}$.

При таком способе построения тоже очевидно, что каждая перестановка из n элементов встречается только один раз. Аналогичные построения можно выполнить, помещая k не справа, а слева либо в любой другой фиксированной позиции.

Если p_n — число перестановок из n объектов, то оба описанных метода показывают, что $p_n = np_{n-1}$; это дает два дополнительных доказательства соотношения $p_n = n(n-1)\ldots(1)$, которое мы уже вывели из (2).

p_n — это очень важная величина, которую называют *n-факториал* и записывают следующим образом:

$$n! = 1\cdot 2\cdot \ldots \cdot n = \prod_{k=1}^{n} k. \tag{4}$$

Из нашего соглашения, касающегося пустых произведений (раздел 1.2.3), следует, что

$$0! = 1. \tag{5}$$

Учитывая это соглашение, видим, что основное тождество

$$n! = (n-1)!\,n \tag{6}$$

справедливо для всех целых положительных n.

В вычислительных задачах факториалы встречаются довольно часто, поэтому я советую читателю запомнить значения первых из них:

$$0! = 1,\quad 1! = 1,\quad 2! = 2,\quad 3! = 6,\quad 4! = 24,\quad 5! = 120.$$

Факториалы растут очень быстро, например 1000! — это целое число, имеющее свыше 2 500 цифр.

Очень полезно запомнить значение $10! = 3\,628\,800$; кроме того, нужно знать, что $10!$ — это примерно $3\frac{1}{2}$ миллиона. В определенном смысле это число представляет собой некую грань между тем, что реально можно вычислить на компьютере, и тем, что нельзя. Если в алгоритме предусмотрена проверка более чем 10! случаев,

то на практике для его выполнения может потребоваться слишком много машинного времени. С другой стороны, если нужно проверить 10! случаев, на каждый из которых требуется, скажем, одна миллисекунда машинного времени, то выполнение алгоритма займет не больше часа. Я согласен, что все эти рассуждения весьма туманны, но во всяком случае они помогут вам получить некоторое представление о том, что реально можно вычислить на компьютере, а что нет.

Естественно, возникает вопрос, как связано $n!$ с другими математическими величинами. Можно ли определить, насколько велико значение 1000!, не выполняя трудоемких операций умножения по формуле (4)? Ответ на этот вопрос дал Джеймс Стирлинг (James Stirling) в своей знаменитой работе *Methodus Differentialis* (1730 г.), с. 137. Он получил формулу

$$n! \approx \sqrt{2\pi n}\left(\frac{n}{e}\right)^n \tag{7}$$

Здесь "$\approx$" означает "приближенно равно", а "e" — это основание натурального логарифма (см. раздел 1.2.2). Приближенную формулу Стирлинга (7) мы докажем в разделе 1.2.11.2. А простое доказательство менее точного результата дано в упр. 24.

Рассмотрим пример применения формулы Стирлинга. Вычислим

$$40320 = 8! \approx 4\sqrt{\pi}\left(\frac{8}{e}\right)^8 = 2^{26}\sqrt{\pi}\,e^{-8} \approx 67108864 \cdot 1.77245 \cdot 0.00033546 \approx 39902.$$

В данном случае погрешность составляет приблизительно 1%; впоследствии мы увидим, что относительная погрешность расчетов по этой формуле приблизительно равна $1/(12n)$.

Помимо приближенного значения, которое вычисляется по формуле (7), можно довольно легко получить точное значение $n!$ с помощью разложения на простые множители. *Действительно, простое число p является делителем $n!$ кратности*

$$\mu = \left\lfloor\frac{n}{p}\right\rfloor + \left\lfloor\frac{n}{p^2}\right\rfloor + \left\lfloor\frac{n}{p^3}\right\rfloor + \cdots = \sum_{k>0}\left\lfloor\frac{n}{p^k}\right\rfloor. \tag{8}$$

Например, если $n = 1000$ и $p = 3$, то имеем

$$\begin{aligned}\mu &= \left\lfloor\frac{1000}{3}\right\rfloor + \left\lfloor\frac{1000}{9}\right\rfloor + \left\lfloor\frac{1000}{27}\right\rfloor + \left\lfloor\frac{1000}{81}\right\rfloor + \left\lfloor\frac{1000}{243}\right\rfloor + \left\lfloor\frac{1000}{729}\right\rfloor \\ &= 333 + 111 + 37 + 12 + 4 + 1 = 498.\end{aligned}$$

Таким образом, 1000! делится на 3^{498}, но не на 3^{499}. Хотя формула (8) записана в виде бесконечной суммы, на самом деле для любых чисел n и p эта сумма имеет конечное число слагаемых; дело в том, что, начиная с некоторого k, все эти слагаемые обращаются в нуль. Из упр. 1.2.4–35 следует, что $\lfloor n/p^{k+1}\rfloor = \lfloor\lfloor n/p^k\rfloor/p\rfloor$; это позволяет упростить вычисления по формуле (8), так как можно просто разделить значение предыдущего члена на p и отбросить остаток.

Формула (8) следует из того, что $\lfloor n/p^k\rfloor$ — это количество целых чисел из $\{1, 2, \ldots, n\}$, кратных p^k. Если рассмотреть целые числа в произведении (4), то станет ясно, что любое целое, которое делится на p^j, но не делится на p^{j+1}, учитывается ровно j раз: один раз — в $\lfloor n/p\rfloor$, другой раз — в $\lfloor n/p^2\rfloor$, ... и, наконец, в $\lfloor n/p^j\rfloor$. Таким образом, учитываются все случаи, когда p является простым множителем в $n!$.

Теперь возникает еще один естественный вопрос. Мы определили $n!$ для неотрицательных целых чисел n. Но, возможно, факториальная функция имеет смысл также для рациональных и даже для действительных чисел n? Например, чему равен $(\frac{1}{2})!$? Для иллюстрации этого вопроса давайте введем "термиальную" функцию

$$n? = 1 + 2 + \cdots + n = \sum_{k=1}^{n} k, \tag{9}$$

которая аналогична факториальной, за исключением того, что здесь выполняется сложение, а не умножение.

Из формулы 1.2.3–(15) мы уже знаем, чему равна сумма арифметической прогрессии:

$$n? = \tfrac{1}{2}n(n+1). \tag{10}$$

Таким образом, используя формулу (10) вместо (9), можно обобщить определение "термиальной" функции для произвольного n. И, отвечая на заданный выше вопрос, получим $(\frac{1}{2})? = \frac{3}{8}$.

Стирлинг сам предпринял несколько попыток обобщить понятие факториала $n!$ для нецелых n. Он представил приближение (7) в виде бесконечной суммы, но, к сожалению, эта сумма не сходилась ни для одного значения n. Формула (7) дает очень хорошее приближение, но ее нельзя продолжить так, чтобы получить *точное* значение. [Эта довольно необычная ситуация обсуждается в книге K. Knopp, *Theory and Application of Infinite Series*, 2nd ed. (Glasgow: Blackie, 1951), 518–520, 527, 534.]

Стирлинг предпринял еще одну попытку, обратив внимание на то, что

$$\begin{aligned} n! = 1 &+ \Big(1 - \frac{1}{1!}\Big)n + \Big(1 - \frac{1}{1!} + \frac{1}{2!}\Big)n(n-1) \\ &+ \Big(1 - \frac{1}{1!} + \frac{1}{2!} - \frac{1}{3!}\Big)n(n-1)(n-2) + \cdots. \end{aligned} \tag{11}$$

(Мы докажем эту формулу в следующем разделе.) Данная сумма из соотношения (11) кажется бесконечной, но на самом деле она конечна для любого неотрицательного целого n. Тем не менее формула (11) не дает желаемого обобщения для функции $n!$, так как бесконечная сумма сходится *только* в том случае, если n — неотрицательное целое число (см. упр. 16).

Но Стирлинг не упал духом и нашел последовательность $a_1, a_2, \ldots$, такую, что

$$\ln n! = a_1 n + a_2 n(n-1) + \cdots = \sum_{k\ge 0} a_{k+1} \prod_{0\le j\le k} (n-j). \tag{12}$$

Однако он не смог *доказать*, что эта сумма определяет функцию $n!$ для всех дробных n, хотя и нашел значение $(\frac{1}{2})! = \sqrt{\pi}/2$.

Приблизительно в то же время данной задачей занимался и Леонард Эйлер (Leonhard Euler), и именно он первым нашел обоснованное обобщение:

$$n! = \lim_{m\to\infty} \frac{m^n m!}{(n+1)(n+2)\ldots(n+m)}. \tag{13}$$

Эйлер изложил эту идею в письме Христиану Гольдбаху (Christian Goldbach) от 13 октября 1729 года. В его формуле $n!$ определяется для любого значения n,

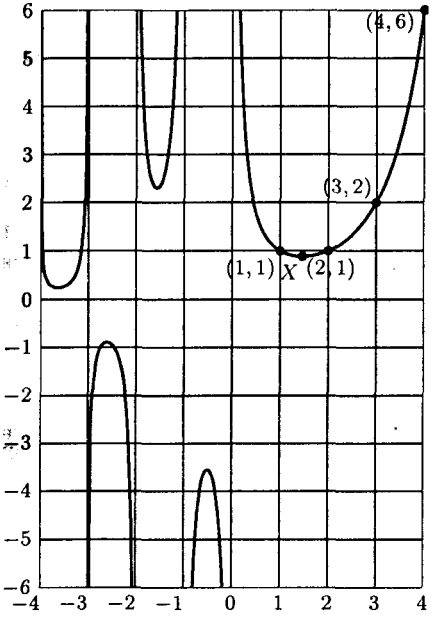


Рис. 7. Функция $\Gamma(x) = (x-1)!$. Локальный минимум в точке X имеет координаты (1.46163 21449 68362 34126 26595, 0.88560 31944 10888 70027 88159).

за исключением целых отрицательных, для которых знаменатель в формуле (11) обращается в нуль; в таких случаях $n!$ считается равным бесконечности. В упр. 8 и 22 объясняется, почему формула (13) является обоснованным разумным определением факториала.

Примерно двумя столетиями позже, в 1900 году, Ш. Эрмит (C. Hermite) доказал, что на самом деле формула Стирлинга (12) корректно определяет $n!$ для нецелых n и что фактически обобщения Эйлера и Стирлинга идентичны.

В прошлые века для факториалов использовались самые разные обозначения. Эйлер писал $[n]$, Гаусс (Gauss) — $\Pi\, n$, а в Англии и Италии были популярны символы $\underline{|n}$ и $\underline{n|}$. В наше время для целых n универсальным обозначением факториала является $n!$; его ввел сравнительно малоизвестный математик Кристиан Крамп (Christian Kramp) в своем учебнике по алгебре [*Élémens d'Arithmétique Universelle* (Cologne: 1808)].

Но для *нецелых* n запись $n!$ не является общепринятой; в этом случае обычно используется обозначение, которое ввел А. М. Лежандр (A. M. Legendre):

$$n! = \Gamma(n+1) = n\Gamma(n). \tag{14}$$

Функция $\Gamma(x)$ называется *гамма-функцией*, и из (13) получаем формулу для нее:

$$\Gamma(x) = \frac{x!}{x} = \lim_{m\to\infty} \frac{m^x m!}{x(x+1)(x+2)\ldots(x+m)}. \tag{15}$$

График функции $\Gamma(x)$ показан на рис. 7.

Формулы (13) и (15) определяют факториалы и гамма-функцию как для действительных, так и для комплексных чисел; но если подразумевается переменная,

имеющая и действительную, и мнимую части, то для ее обозначения обычно используется буква z, а не n или x. Связь между факториалом и гамма-функцией выражается не только формулой $z! = \Gamma(z+1)$, но и соотношением

$$(-z)!\,\Gamma(z) = \frac{\pi}{\sin \pi z}, \tag{16}$$

которое выполняется для всех нецелых z (см. упр. 23).

Хотя $\Gamma(z)$ равна бесконечности для отрицательного целого или равного нулю z, функция $1/\Gamma(z)$ корректно определяется для всех комплексных z (см. упр. 1.2.7–24). Применяя гамма-функцию в теоретических исследованиях, часто приходится пользоваться интегральным представлением Германа Ганкеля (Hermann Hankel):

$$\frac{1}{\Gamma(z)} = \frac{1}{2\pi i} \oint \frac{e^t\,dt}{t^z}; \tag{17}$$

причем путь интегрирования в комплексной плоскости идет из $-\infty$ по отрицательной части действительной оси, затем огибает начало координат в положительном направлении (против часовой стрелки) и опять по отрицательной части оси X возвращается в $-\infty$. [*Zeitschrift für Math. und Physik* **9** (1864), 1–21.]

Во многих формулах дискретной математики используются произведения факториального типа, которые называются *факториальными степенями*. Для положительного целого k величины $x^{\underline{k}}$ и $x^{\overline{k}}$ определяются следующим образом:

$$x^{\underline{k}} = x(x-1)\ldots(x-k+1) = \prod_{j=0}^{k-1}(x-j); \tag{18}$$

$$x^{\overline{k}} = x(x+1)\ldots(x+k-1) = \prod_{j=0}^{k-1}(x+j). \tag{19}$$

Так, например, величина p_{nk} из (2) — это просто $n^{\underline{k}}$. Заметим, что

$$x^{\overline{k}} = (x+k-1)^{\underline{k}} = (-1)^k(-x)^{\underline{k}}. \tag{20}$$

Общие формулы

$$x^{\underline{k}} = \frac{x!}{(x-k)!}, \qquad x^{\overline{k}} = \frac{\Gamma(x+k)}{\Gamma(x)} \tag{21}$$

можно использовать для определения факториальных степеней для не целых значений k. [Обозначение $x^{\overline{k}}$ ввел А. Капелли (A. Capelli) в работе *Giornale di Matematiche di Battaglini* **31** (1893), 291–313.]

Захватывающая история изучения факториалов со времен Стирлинга до наших дней прослежена в статье P. J. Davis "Leonhard Euler's integral: A historical profile of the gamma function" *AMM* **66** (1959), 849–869; см. также работу J. Dutka, *Archive for History of Exact Sciences* **31** (1984), 15–34.

УПРАЖНЕНИЯ

1. [*00*] Сколькими способами можно перетасовать колоду из 52 карт?

2. [*10*] Используя обозначения из соотношения (2), покажите, что $p_{n(n-1)} = p_{nn}$, и объясните, почему это так.

3. [*10*] Какие перестановки чисел $\{1,2,3,4,5\}$ можно построить из перестановки 3 1 2 4 с помощью методов 1 и 2 соответственно?

▶ **4.** [*13*] Учитывая тот факт, что $\log_{10} 1000! = 2567.60464\ldots$, определите точно, сколько цифр содержится в числе 1000!. Какая цифра стоит в *старшем* разряде? Какая в *младшем*?

5. [*15*] Оцените значение 8! с помощью следующей более точной приближенной формулы Стирлинга:

$$n! \approx \sqrt{2\pi n}\left(\frac{n}{e}\right)^n\left(1+\frac{1}{12n}\right).$$

▶ **6.** [*17*] Используя формулу (8), запишите 20! в виде произведения простых сомножителей.

7. [*M10*] Покажите, что "обобщенная термиальная" функция, определенная формулой (10), удовлетворяет тождеству $x? = x + (x-1)?$ для всех действительных x.

8. [*HM15*] Покажите, что предел в формуле (13) равен $n!$ и в том случае, если n — неотрицательное целое число.

9. [*M10*] Определите значения $\Gamma(\frac{1}{2})$ и $\Gamma(-\frac{1}{2})$ на основании того, что $(\frac{1}{2})! = \sqrt{\pi}/2$.

▶ **10.** [*HM20*] Справедливо ли тождество $\Gamma(x+1) = x\Gamma(x)$ для всех действительных чисел x (см. упр. 7)?

11. [*M15*] Пусть представление числа n в двоичной системе выглядит следующим образом: $n = 2^{e_1} + 2^{e_2} + \cdots + 2^{e_r}$, где $e_1 > e_2 > \cdots > e_r \geq 0$. Покажите, что $n!$ делится на 2^{n-r} но не делится на 2^{n-r+1}.

▶ **12.** [*M22*] (А. Лежандр, 1808.) Обобщим результат предыдущего упражнения. Пусть p — простое число и представление n в системе счисления с основанием p имеет вид $n = a_k p^k + a_{k-1}p^{k-1} + \cdots + a_1 p + a_0$. Найдите простую формулу, выражающую число μ из формулы (8) через n, p и коэффициенты a_k.

13. [*M23*] (*Теорема Вильсона (Wilson)*, на самом деле доказанная Лейбницем в 1682 г.) Если p — простое число, то $(p-1)! \bmod p = p-1$. Докажите это, разбивая элементы из множества $\{1,2,\ldots,p-1\}$ на пары таких чисел, произведение которых по модулю p равно 1.

▶ **14.** [*M28*] (Л. Штикельбергер (L. Stickelberger), 1890.) С помощью обозначений из упр. 12 можно определить $n! \bmod p$ в виде представления в системе счисления с основанием p для *любого* положительного целого n, тем самым обобщив теорему Вильсона. Докажите, что $n!/p^\mu \equiv (-1)^\mu a_0! a_1! \ldots a_k!$ (по модулю p).

15. [*HM15*] *Перманент* квадратной матрицы вычисляется по той же формуле, что и определитель, но каждому члену этой формулы присваивается знак "плюс", в то время как в формуле определителя чередуются знаки "плюс" и "минус". Таким образом, перманент матрицы

$$\begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix}$$

равен $aei + bfg + cdh + gec + hfa + idb$. Чему равен перманент матрицы

$$\begin{pmatrix} 1\times 1 & 1\times 2 & \ldots & 1\times n \\ 2\times 1 & 2\times 2 & \ldots & 2\times n \\ \vdots & \vdots & \ddots & \vdots \\ n\times 1 & n\times 2 & \ldots & n\times n \end{pmatrix}?$$

16. [*HM15*] Покажите, что бесконечная сумма в (11) расходится, если n не является неотрицательным целым числом.

17. [*HM20*] Докажите, что бесконечное произведение

$$\prod_{n\geq 1}\frac{(n+\alpha_1)\ldots(n+\alpha_k)}{(n+\beta_1)\ldots(n+\beta_k)}$$

равно $\Gamma(1+\beta_1)\ldots\Gamma(1+\beta_k)/\Gamma(1+\alpha_1)\ldots\Gamma(1+\alpha_k)$, если $\alpha_1+\cdots+\alpha_k=\beta_1+\cdots+\beta_k$ и ни одно из β_i не является отрицательным целым числом.

18. [*M20*] Предположим, что $\pi/2=\frac{2}{1}\cdot\frac{2}{3}\cdot\frac{4}{3}\cdot\frac{4}{5}\cdot\frac{6}{5}\cdot\frac{6}{7}\cdot\cdots$. (Это "произведение Валлиса", полученное Дж. Валлисом (J. Wallis) в 1655 году; мы докажем его в упр. 1.2.6–43.) Используя предыдущее упражнение, докажите, что $(\frac{1}{2})!=\sqrt{\pi}/2$.

19. [*HM22*] Обозначим величину, стоящую после "$\lim_{m\to\infty}$" в формуле (15), через $\Gamma_m(x)$. Покажите, что

$$\Gamma_m(x)=\int_0^m\Big(1-\frac{t}{m}\Big)^m t^{x-1}\,dt=m^x\int_0^1(1-t)^m t^{x-1}\,dt,\quad\text{если } x>0.$$

20. [*HM21*] Учитывая тот факт, что $0\leq e^{-t}-(1-t/m)^m\leq t^2e^{-t}/m$, где $0\leq t\leq m$, и предыдущее упражнение, покажите, что $\Gamma(x)=\int_0^\infty e^{-t}t^{x-1}\,dt$ при $x>0$.

21. [*HM25*] (Л. Ф. А. Арбогаст (L. F. A. Arbogast), 1800.) Пусть $D_x^k u$ — это k-я производная функции u по x. По правилу дифференцирования сложной функции $D_x^1 w=D_u^1 w\,D_x^1 u$. Применяя это правило при нахождении второй производной, получим $D_x^2 w=D_u^2 w(D_x^1 u)^2+D_u^1 w\,D_x^2 u$. Покажите, что *общая формула* для производной n-го порядка имеет вид

$$D_x^n w=\sum_{j=0}^{n}\sum_{\substack{k_1+k_2+\cdots+k_n=j\\ k_1+2k_2+\cdots+nk_n=n\\ k_1,k_2,\ldots,k_n\geq 0}} D_u^j w\,\frac{n!}{k_1!\,(1!)^{k_1}\ldots k_n!\,(n!)^{k_n}}(D_x^1u)^{k_1}\ldots(D_x^n u)^{k_n}.$$

▸ **22.** [*HM20*] Попробуйте поставить себя на место Эйлера, когда он искал способ обобщения понятия факториала $n!$ для нецелых значений n. Так как $(n+\frac{1}{2})!/n!$ умножить на $((n+\frac{1}{2})+\frac{1}{2})!/(n+\frac{1}{2})!$ равно $(n+1)!/n!=n+1$, кажется естественным, что $(n+\frac{1}{2})!/n!$ должно приближенно равняться $\sqrt{n}$. Аналогично $(n+\frac{1}{3})!/n!$ должно приближенно равняться $\approx\sqrt[3]{n}$. Выдвиньте гипотезу о поведении отношения $(n+x)!/n!$, когда n стремится к бесконечности. Будет ли ваша гипотеза справедлива для целых x? Можно ли с ее помощью найти приближенное значение $x!$ для нецелого x?

23. [*HM20*] Докажите формулу (16), исходя из того, что $\pi z\prod_{n=1}^\infty(1-z^2/n^2)=\sin\pi z$.

▸ **24.** [*HM21*] Докажите полезные неравенства

$$\frac{n^n}{e^{n-1}}\leq n!\leq\frac{n^{n+1}}{e^{n-1}},\qquad\text{где } n\text{ — целое, } n\geq 1.$$

[*Указание.* $1+x\leq e^x$ для всех действительных x, поэтому $(k+1)/k\leq e^{1/k}\leq k/(k-1)$.]

25. [*M20*] Выполняется ли для факториальных степеней правило, аналогичное обычному правилу сложения показателей степеней $x^{m+n}=x^m x^n$?

1.2.6. Биномиальные коэффициенты

Сочетания из n объектов по k — это возможные варианты выбора k различных элементов из n объектов без учета порядка расположения этих элементов. Приведем пример сочетаний из пяти объектов $\{a,b,c,d,e\}$ по три:

$$abc,\quad abd,\quad abe,\quad acd,\quad ace,\quad ade,\quad bcd,\quad bce,\quad bde,\quad cde. \qquad (1)$$

Таблица 1

ТАБЛИЦА БИНОМИАЛЬНЫХ КОЭФФИЦИЕНТОВ (ТРЕУГОЛЬНИК ПАСКАЛЯ)

r	$\binom{r}{0}$	$\binom{r}{1}$	$\binom{r}{2}$	$\binom{r}{3}$	$\binom{r}{4}$	$\binom{r}{5}$	$\binom{r}{6}$	$\binom{r}{7}$	$\binom{r}{8}$	$\binom{r}{9}$
0	1	0	0	0	0	0	0	0	0	0
1	1	1	0	0	0	0	0	0	0	0
2	1	2	1	0	0	0	0	0	0	0
3	1	3	3	1	0	0	0	0	0	0
4	1	4	6	4	1	0	0	0	0	0
5	1	5	10	10	5	1	0	0	0	0
6	1	6	15	20	15	6	1	0	0	0
7	1	7	21	35	35	21	7	1	0	0
8	1	8	28	56	70	56	28	8	1	0
9	1	9	36	84	126	126	84	36	9	1

Подсчитать общее число сочетаний из n объектов по k совсем несложно. Соотношение (2) из предыдущего раздела говорит о том, что существует $n(n-1)\dots(n-k+1)$ способов выбора первых k объектов в перестановке, причем каждое сочетание из k элементов встречается ровно $k!$ раз, так как для любого набора из k объектов существует $k!$ перестановок. Поэтому для числа сочетаний из n элементов по k, которое мы обозначим через $\binom{n}{k}$, получаем следующую формулу:

$$\binom{n}{k} = \frac{n(n-1)\dots(n-k+1)}{k(k-1)\dots(1)}. \tag{2}$$

Например,

$$\binom{5}{3} = \frac{5\cdot 4\cdot 3}{3\cdot 2\cdot 1} = 10.$$

Это число сочетаний, которые мы нашли в (1).

Величины $\binom{n}{k}$ (читается "число сочетаний из n по k") называются *биномиальными коэффициентами* и имеют очень широкое применение. Вероятно, это самые важные величины, использующиеся в анализе алгоритмов, поэтому я настоятельно советую читателю познакомиться с ними.

Формулу (2) можно использовать для определения $\binom{n}{k}$ даже в том случае, когда n *не является целым числом*. Точнее говоря, определим число сочетаний $\binom{r}{k}$ для всех действительных r и всех целых k следующим образом:

$$\binom{r}{k} = \frac{r(r-1)\dots(r-k+1)}{k(k-1)\dots(1)} = \frac{r^{\underline{k}}}{k!} = \prod_{j=1}^{k}\frac{r+1-j}{j}, \qquad \text{целое } k \geq 0;$$

$$\binom{r}{k} = 0, \qquad \text{целое } k < 0. \tag{3}$$

В частных случаях имеем:

$$\binom{r}{0} = 1, \qquad \binom{r}{1} = r, \qquad \binom{r}{2} = \frac{r(r-1)}{2}. \tag{4}$$

В табл. 1 приведены значения биномиальных коэффициентов для небольших целых величин r и k; биномиальные коэффициенты для $0 \leq r \leq 4$ следует запомнить.

Биномиальные коэффициенты имеют продолжительную и интересную историю. Табл. 1 называется треугольником Паскаля, потому что она была приведена в работе Блеза Паскаля *Traité du Triangle Arithmétique*, вышедшей в 1653 году. Этот трактат имел очень важное значение, так как он представлял собой одну из первых работ по теории вероятностей; но биномиальные коэффициенты были открыты не Паскалем (к этому времени они были уже хорошо известны в Европе). Табл. 1 приведена также в трактате *Сы юань юй цзянь* ("Яшмовое зеркало четырех элементов"), написанном китайским математиком Чжу Ши-Цзе (Shih-Chieh Chu) в 1303 году. В нем говорится, что биномиальные коэффициенты известны с давних времен. Самое раннее (из известных) подробное описание биномиальных коэффициентов встречается в комментарии, написанном в 10 веке Халаюдхой (Halāyudha) к произведению древнеиндийской классики *Чанда-сутра* Пингала. [См. Г. Чакраварти (G. Chakravarti), *Bull. Calcutta Math. Soc.* **24** (1932), 79–88.] Примерно в 1150 году индийский математик Бхаскара Ачарья (Bhāskara Āchārya) в книге *Лилавати*, ч. 6, гл. 4, дал очень четкое описание биномиальных коэффициентов. Для малых значений k эти коэффициенты были известны намного раньше; упоминание о них вместе с геометрической интерпретацией встречалось в работах греческих и римских авторов (рис. 8). Обозначение $\binom{r}{k}$ было введено Андреасом фон Эттингсхаузеном (Andreas von Ettingshausen) в книге *Die combinatorische Analysis* (Vienna, 1826).

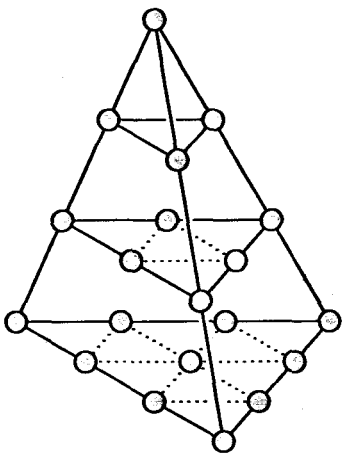


Рис. 8. Геометрическая интерпретация $\binom{n+2}{3}$, $n = 4$.

Читатель, вероятно, уже заметил в табл. 1 несколько интересных закономерностей. Существуют буквально тысячи тождеств, в которых содержатся биномиальные коэффициенты, и на протяжении многих веков не прекращалось изучение их замечательных свойств. Соотношений, в которых используются биномиальные коэффициенты, так много, что открытие нового тождества не волнует практически никого, кроме самого исследователя. Для работы с формулами, использующимися в анализе алгоритмов, умение обращаться с биномиальными коэффициентами является совершенно необходимым, поэтому в данном разделе я попытаюсь в доступной форме объяснить, как это делается. Марк Твен однажды попытался свести все шутки примерно к десяти основным типам (о дочери фермера, теще и т. д.), а мы попробуем свести тысячи тождеств к небольшому набору основных операций, позволяющих решить практически любую задачу, в которой фигурируют биномиальные коэффициенты.

Числа r и k из $\binom{r}{k}$ в большинстве случаев будут целыми. Заметим, что отдельные методы, о которых пойдет речь, применимы только в подобной ситуации. Поэтому справа от каждого пронумерованного соотношения будем подробно перечислять все имеющиеся ограничения на переменные. Например, в соотношении (3) указано, что k — целое; в то же время для r никаких ограничений нет. Заметим, что самыми полезными являются те тождества, которые справедливы при наименьшем числе ограничений.

А теперь давайте изучим основные методы работы с биномиальными коэффициентами.

A. Факториальное представление. Из соотношения (3) непосредственно получаем

$$\binom{n}{k} = \frac{n!}{k!\,(n-k)!}, \quad \text{целое } n \geq \text{целого } k \geq 0. \tag{5}$$

Эта формула позволяет представлять комбинации факториалов в виде биномиальных коэффициентов, и наоборот.

B. Свойство симметрии. Из соотношений (3) и (5) получаем

$$\binom{n}{k} = \binom{n}{n-k}, \quad \text{целое } n \geq 0, \text{целое } k. \tag{6}$$

Эта формула справедлива для всех целых k. *Если k отрицательно или больше, чем n, то биномиальный коэффициент равен нулю* (при условии, что n — неотрицательное целое число).

C. Внесение-вынесение. В силу определения (3) имеем

$$\binom{r}{k} = \frac{r}{k}\binom{r-1}{k-1}, \quad \text{целое } k \neq 0. \tag{7}$$

Эта формула очень полезна для комбинирования биномиального коэффициента с другими частями выражения. Выполнив элементарные преобразования, получаем правила

$$k\binom{r}{k} = r\binom{r-1}{k-1}, \quad \frac{1}{r}\binom{r}{k} = \frac{1}{k}\binom{r-1}{k-1},$$

причем первое из них справедливо для всех целых k, а второе — если k и r не равны нулю. Мы имеем также еще одно аналогичное соотношение:

$$\binom{r}{k} = \frac{r}{r-k}\binom{r-1}{k}, \quad \text{целое } k \neq r. \tag{8}$$

Давайте продемонстрируем эти преобразования на примере доказательства формулы (8), используя поочередно формулы (6) и (7):

$$\binom{r}{k} = \binom{r}{r-k} = \frac{r}{r-k}\binom{r-1}{r-1-k} = \frac{r}{r-k}\binom{r-1}{k}.$$

[*Замечание.* Данное доказательство имеет силу только в случае, когда r — положительное целое число $\neq k$, так как этого требуют ограничения, наложенные на соотношения (6) и (7). Утверждается, что формула (8) справедлива для *произвольного* $r \neq k$. Доказать это можно с помощью простого, но важного приема. Мы убедились в том, что

$$r\binom{r-1}{k} = (r-k)\binom{r}{k}$$

для *бесконечного множества значений* r. Обе части этого равенства являются *многочленами* от r. Ненулевой многочлен степени n может иметь максимум n различных корней. Поэтому, если *два многочлена степени* $\leq n$ *совпадают в* $n+1$ *или более различных точках, то, вычитая их один из другого, получим, что эти*

многочлены тождественно равны. Данный принцип можно использовать для доказательства того, что многие тождества, верные для целых чисел, справедливы для всех действительных чисел.]

D. Формула сложения. Очевидно, что основное соотношение

$$\binom{r}{k} = \binom{r-1}{k} + \binom{r-1}{k-1}, \quad \text{целое } k, \tag{9}$$

выполняется для табл. 1 (каждое значение равно сумме двух значений из предыдущего ряда, причем одно находится в том же столбце, а другое — в ближайшем столбце слева) и его можно легко вывести из соотношения (3). Но есть и другой способ. Из формул (7) и (8) получаем

$$r\binom{r-1}{k} + r\binom{r-1}{k-1} = (r-k)\binom{r}{k} + k\binom{r}{k} = r\binom{r}{k}.$$

Формула (9) часто используется в доказательствах индукцией по r, когда r — целое число.

E. Формулы суммирования. Повторное применение формулы (9) дает

$$\binom{r}{k} = \binom{r-1}{k} + \binom{r-1}{k-1} = \binom{r-1}{k} + \binom{r-2}{k-1} + \binom{r-2}{k-2} = \cdots$$

или

$$\binom{r}{k} = \binom{r-1}{k-1} + \binom{r-1}{k} = \binom{r-1}{k-1} + \binom{r-2}{k-1} + \binom{r-2}{k} = \cdots.$$

Таким образом, получаем две важные формулы суммирования, которые можно выразить следующим образом:

$$\sum_{k=0}^{n}\binom{r+k}{k} = \binom{r}{0} + \binom{r+1}{1} + \cdots + \binom{r+n}{n} = \binom{r+n+1}{n},$$

$$\text{целое } n \geq 0; \tag{10}$$

$$\sum_{k=0}^{n}\binom{k}{m} = \binom{0}{m} + \binom{1}{m} + \cdots + \binom{n}{m} = \binom{n+1}{m+1}$$

$$\text{целое } m \geq 0, \text{ целое } n \geq 0. \tag{11}$$

Формулу (11) можно легко доказать индукцией по n, но давайте посмотрим, как вывести ее из формулы (10) в результате двукратного применения соотношения (6),

$$\sum_{0\leq k\leq n}\binom{k}{m} = \sum_{0\leq m+k\leq n}\binom{m+k}{m} = \sum_{-m\leq k<0}\binom{m+k}{m} + \sum_{0\leq k\leq n-m}\binom{m+k}{k}$$

$$= 0 + \binom{m+(n-m)+1}{n-m} = \binom{n+1}{m+1},$$

в предположении, что $n \geq m$. Если $n < m$, то (11) очевидно.

Формула (11) применяется очень часто; фактически мы уже вывели ее частные случаи в предыдущих разделах. Например, при $m = 1$ получаем старую добрую

формулу суммы арифметической прогрессии:

$$\binom{0}{1}+\binom{1}{1}+\cdots+\binom{n}{1}=0+1+\cdots+n=\binom{n+1}{2}=\frac{(n+1)n}{2}.$$

Предположим, нам нужна простая формула для суммы $1^2+2^2+\cdots+n^2$. Ее можно вывести, если заметить, что $k^2=2\binom{k}{2}+\binom{k}{1}$; отсюда

$$\sum_{k=0}^{n}k^2=\sum_{k=0}^{n}\left(2\binom{k}{2}+\binom{k}{1}\right)=2\binom{n+1}{3}+\binom{n+1}{2}.$$

Эту формулу, выраженную через биномиальные коэффициенты, при желании можно снова записать в виде многочлена:

$$1^2+2^2+\cdots+n^2=2\frac{(n+1)n(n-1)}{6}+\frac{(n+1)n}{2}=\tfrac{1}{3}n(n+\tfrac{1}{2})(n+1). \qquad (12)$$

Аналогично можно получить формулу для суммы $1^3+2^3+\cdots+n^3$; и вообще, *любой* многочлен типа $a_0+a_1k+a_2k^2+\cdots+a_mk^m$ можно представить в виде $b_0\binom{k}{0}+b_1\binom{k}{1}+\cdots+b_m\binom{k}{m}$, где $b_0,\ldots,b_m$ — соответствующим образом подобранные коэффициенты. Мы вернемся к этому вопросу несколько позже.

F. Биномиальная теорема. Сформулируем биномиальную теорему, которая, без сомнения, является одним из наших главных инструментов:

$$(x+y)^r=\sum_k\binom{r}{k}x^ky^{r-k},\quad \text{целое } r\ge 0. \qquad (13)$$

Например, $(x+y)^4=x^4+4x^3y+6x^2y^2+4xy^3+y^4$. (Наконец-то мы можем на законных основаниях использовать термин "биномиальные коэффициенты" для чисел $\binom{r}{k}$!)

Важно отметить, что в формуле (13) мы записали сумму вида $\sum_k$, а не $\sum_{k=0}^{r}$, как можно было ожидать. Если на k не наложено никаких ограничений, то суммирование производится по *всем* целым k, $-\infty<k<+\infty$. В данном случае две приведенные записи эквивалентны, так как для $k<0$ или $k>r$ соответствующие члены суммы из формулы (13) обращаются в нуль. Но форма записи $\sum_k$ более предпочтительна, так как все операции с суммами выполняются проще, когда условия суммирования являются более простыми. Мы существенно сэкономим время и силы, если не будем следить за верхним и/или нижним пределом суммирования; поэтому, по возможности, данные пределы имеет смысл оставлять неопределенными. Выбранный нами тип записи имеет еще одно преимущество: если r не является неотрицательным целым числом, то сумма в формуле (13) становится *бесконечной* и *биномиальная теорема* математического анализа утверждает, что *соотношение* (13) *справедливо для всех* r, если $|x/y|<1$.

Следует отметить, что формула (13) не противоречит равенству

$$0^0=1, \qquad (14)$$

которое принимается по определению. Мы везде будем следовать этому соглашению.

Частный случай формулы (13) для $y = 1$ настолько важен, что сформулируем его отдельно:

$$\sum_k \binom{r}{k} x^k = (1+x)^r, \qquad \text{целое } r \geq 0 \text{ или } |x| < 1. \tag{15}$$

Об открытии биномиальной теоремы Исаак Ньютон объявил в письмах к Ольденбургу (Oldenburg) от 13 июня и 24 октября 1676 года. [См. D. Struik, *Source Book in Mathematics* (Harvard Univ. Press, 1969), 284–291.] Но, по всей видимости, у него не было настоящего доказательства формулы бинома; в те времена исследователи еще не вполне осознавали необходимость строгого доказательства теорем. Леонард Эйлер первым попытался доказать эту формулу в 1774 году, но не довел дело до конца. И наконец в 1812 году Карл Фридрих Гаусс дал первое настоящее доказательство биномиальной теоремы. Работа Гаусса представляла собой фактически первый случай, когда было дано удовлетворительное доказательство *чему-либо*, связанному с бесконечными суммами.

В начале 19 века Нильс Хенрик Абель (N. H. Abel) нашел удивительное обобщение формулы бинома (13):

$$(x+y)^n = \sum_k \binom{n}{k} x(x-kz)^{k-1}(y+kz)^{n-k}, \quad \text{целое } n \geq 0,\ x \neq 0. \tag{16}$$

Это тождество по *трем* переменным, x, y и z (см. упр. 50–52). Абель опубликовал и доказал данную формулу в первом томе вскоре ставшего знаменитым журнала А. Л. Крелля (A. L. Crelle) *Journal für die reine und angewandte Mathematik* (1826), 159–160. Интересно заметить, что Абель поместил в том же первом томе много других своих работ, включая известную статью о неразрешимости алгебраических уравнений пятой и более высоких степеней в радикалах, а также о биномиальной теореме. Многочисленные ссылки в связи с формулой (16) можно найти в статье H. W. Gould, *AMM* **69** (1962), 572.

G. Обращение верхнего индекса. Основное тождество

$$\binom{r}{k} = (-1)^k \binom{k-r-1}{k}, \quad \text{целое } k, \tag{17}$$

непосредственно следует из определения (3), если каждый член числителя взять с противоположным знаком и умножить на (-1). Такое преобразование верхнего индекса используется довольно часто.

Простым следствием соотношения (17) является формула суммирования

$$\sum_{k\leq n} \binom{r}{k}(-1)^k = \binom{r}{0} - \binom{r}{1} + \cdots + (-1)^n \binom{r}{n} = (-1)^n \binom{r-1}{n}, \quad \text{целое } n. \tag{18}$$

Это тождество можно доказать по индукции с помощью формулы (9), но мы непосредственно воспользуемся соотношениями (17) и (10):

$$\sum_{k\leq n} \binom{r}{k}(-1)^k = \sum_{k\leq n} \binom{k-r-1}{k} = \binom{-r+n}{n} = (-1)^n \binom{r-1}{n}.$$

Для целого r можно получить еще одно важное следствие формулы (17):

$$\binom{n}{m} = (-1)^{n-m}\binom{-(m+1)}{n-m}, \quad \text{целое } n \geq 0, \text{целое } m. \tag{19}$$

(Положите в (17) $r = n$ и $k = n-m$ и воспользуйтесь формулой (6).) Таким образом, мы переместили n из верхней позиции в нижнюю.

H. Упрощение произведений. Произведения биномиальных коэффициентов можно выразить несколькими различными способами, расписывая их через факториалы по формуле (5) и снова возвращаясь к записи для биномиальных коэффициентов. Например,

$$\binom{r}{m}\binom{m}{k} = \binom{r}{k}\binom{r-k}{m-k}, \quad \text{целое } m, \quad \text{целое } k. \tag{20}$$

Формулу (20) достаточно доказать для r — целого $\geq m$ (см. примечания после формулы (8)) и $0 \leq k \leq m$. Тогда

$$\binom{r}{m}\binom{m}{k} = \frac{r!\,m!}{m!\,(r-m)!\,k!\,(m-k)!} = \frac{r!\,(r-k)!}{k!\,(r-k)!\,(m-k)!\,(r-m)!} = \binom{r}{k}\binom{r-k}{m-k}.$$

Формула (20) очень полезна в случаях, когда индекс (а именно — m) находится и в верхней, и в нижней позициях, а нам нужно, чтобы он был только в одном месте. Обратите внимание, что (7) является частным случаем (20) при $k = 1$.

I. Суммы произведений. Чтобы набор операций с биномиальными коэффициентами был полным, приведем следующие общие тождества, которые доказываются в конце данного раздела. Эти формулы показывают, чему равны суммы произведения двух биномиальных коэффициентов при различных положениях индекса суммирования k:

$$\sum_k \binom{r}{k}\binom{s}{n-k} = \binom{r+s}{n}, \quad \text{целое } n; \tag{21}$$

$$\sum_k \binom{r}{m+k}\binom{s}{n+k} = \binom{r+s}{r-m+n}$$

$$\text{целое } m, \text{целое } n, \text{целое } r \geq 0; \tag{22}$$

$$\sum_k \binom{r}{k}\binom{s+k}{n}(-1)^{r-k} = \binom{s}{n-r}, \quad \text{целое } n, \text{целое } r \geq 0; \tag{23}$$

$$\sum_{k=0}^{r} \binom{r-k}{m}\binom{s}{k-t}(-1)^{k-t} = \binom{r-t-s}{r-t-m},$$

$$\text{целое } t \geq 0, \text{целое } r \geq 0, \text{целое } m \geq 0; \tag{24}$$

$$\sum_{k=0}^{r} \binom{r-k}{m}\binom{s+k}{n} = \binom{r+s+1}{m+n+1}$$

$$\text{целое } n \geq \text{целое } s \geq 0, \text{целое } m \geq 0, \text{целое } r \geq 0; \tag{25}$$

$$\sum_{k\ge 0}\binom{r-tk}{k}\binom{s-t(n-k)}{n-k}\frac{r}{r-tk}=\binom{r+s-tn}{n},\quad \text{целое } n. \tag{26}$$

Самым важным из этих тождеств является соотношение (21). Чтобы облегчить его запоминание, можно интерпретировать правую часть как количество способов выбора n человек среди r мужчин и s женщин, а каждый член суммы слева — как количество способов выбора k мужчин и $n-k$ женщин. Тождество (21) обычно называют сверткой Вандермонда, поскольку А. Вандермонд (A. Vandermonde) опубликовал его в журнале *Mém. Acad. Roy. Sciences Paris* (1772), 489–498. Но на самом деле это тождество уже было приведено в трактате Чжу Ши-Цзе от 1303 года, о котором упоминалось выше [см. J. Needham, *Science and Civilization in China* **3** (Cambridge University Press, 1959), 138–139].

Если в тождестве (26) $r=tk$, то мы избегаем появления нуля в знаменателе, сокращая на $(r-tk)$ и числитель, и знаменатель. Поэтому (26) является полиномиальным тождеством от переменных r, s, t. Очевидно, что (21) — это частный случай (26) при $t=0$.

Следует отметить не совсем очевидные способы применения тождеств (23) и (25). Часто бывает полезно заменить простой биномиальный коэффициент в правой части тождества более сложным выражением из левой части, изменить порядок суммирования и упростить его. Левые части тождеств можно рассматривать как разложения

$$\binom{s}{n+a}\quad\text{по}\quad\binom{s+k}{n}.$$

Тождество (23) используется для отрицательных a, а формула (25) — для положительных a.

На этом наше изучение "науки о биномиальных коэффициентах" заканчивается. Советую читателю внимательно разобраться в приведенных формулах; особенно это касается тождеств (5)–(7), (9), (13), (17), (20) и (21). Обведите их своим любимым маркером!

Имея в своем распоряжении все эти методы, мы сможем решить практически любую возникшую задачу по меньшей мере тремя различными способами. Приведем несколько примеров.

Пример 1. Чему равна сумма $\sum_k\binom{r}{k}\binom{s}{k}k$, где r — положительное целое число?

Решение. Формула (7) позволяет избавиться от "внешнего" k:

$$\sum_k\binom{r}{k}\binom{s}{k}k=\sum_k\binom{r}{k}\binom{s-1}{k-1}s=s\sum_k\binom{r}{k}\binom{s-1}{k-1}$$

Теперь применим формулу (22) при $n=-1$. В результате получим

$$\sum_k\binom{r}{k}\binom{s}{k}k=\binom{r+s-1}{r-1}s,\quad\text{целое } r\ge 0.$$

Пример 2. Чему равна сумма $\sum_k \binom{n+k}{2k}\binom{2k}{k}\frac{(-1)^k}{k+1}$, если n — неотрицательное целое число?

Решение. Это более трудная задача, так как индекс суммирования k встречается в шести местах! Сначала применим формулу (20) и получим

$$\sum_k \binom{n+k}{k}\binom{n}{k}\frac{(-1)^k}{k+1}.$$

Теперь можно вздохнуть с облегчением, поскольку некоторых опасностей, таящихся в первоначальной формуле, мы уже избежали. Следующий шаг очевиден: применим формулу (7) так, как в примере 1:

$$\sum_k \binom{n+k}{k}\binom{n+1}{k+1}\frac{(-1)^k}{n+1} \tag{27}$$

Прекрасно! Исчезло еще одно k. И с этого момента перед нами открываются два одинаково перспективных пути. Можно заменить $\binom{n+k}{k}$ коэффициентом $\binom{n+k}{n}$ в предположении, что $k \geq 0$, и вычислить сумму с помощью формулы (23):

$$\begin{aligned}
&\sum_{k\geq 0}\binom{n+k}{n}\binom{n+1}{k+1}\frac{(-1)^k}{n+1} \\
&\quad = -\frac{1}{n+1}\sum_{k\geq 1}\binom{n-1+k}{n}\binom{n+1}{k}(-1)^k \\
&\quad = -\frac{1}{n+1}\sum_{k\geq 0}\binom{n-1+k}{n}\binom{n+1}{k}(-1)^k + \frac{1}{n+1}\binom{n-1}{n} \\
&\quad = -\frac{1}{n+1}(-1)^{n+1}\binom{n-1}{-1} + \frac{1}{n+1}\binom{n-1}{n} = \frac{1}{n+1}\binom{n-1}{n}.
\end{aligned}$$

Биномиальный коэффициент $\binom{n-1}{n}$ равен нулю, за исключением случая $n = 0$, когда он равен единице. Поэтому решение задачи можно дать в виде $[n=0]$, воспользовавшись обозначением Айверсона (формула 1.2.3–(16)), или в виде δ_{n0}, воспользовавшись символом Кронекера (формула 1.2.3–(19)).

Другой способ решения, начиная от формулы (27), заключается в использовании формулы (17). В результате получаем

$$\sum_k \binom{-(n+1)}{k}\binom{n+1}{k+1}\frac{1}{n+1}.$$

Теперь можно применить формулу (22) и получить

$$\binom{n+1-(n+1)}{n+1-1+0}\frac{1}{n+1} = \binom{0}{n}\frac{1}{n+1}.$$

Итак, мы снова получили тот же ответ:

$$\sum_k \binom{n+k}{2k}\binom{2k}{k}\frac{(-1)^k}{k+1} = \delta_{n0}, \qquad \text{целое } n \geq 0. \tag{28}$$

Пример 3. Чему равна сумма $\sum_k \binom{n+k}{m+2k}\binom{2k}{k}\frac{(-1)^k}{k+1}$, если m и n — положительные целые числа?

Решение. Если бы m было равно нулю, мы получили бы ту же сумму с которой имели дело в примере 2. Но присутствие ненулевого m означает, что мы не можем воспользоваться методом из предыдущего примера, так как уже на первом шаге формулу (20) применить нельзя. В этой ситуации имеет смысл усложнить задачу, заменив нежелательный коэффициент $\binom{n+k}{m+2k}$ суммой членов вида $\binom{x+k}{2k}$. В результате задача сведется к ряду задач, решать которые мы уже умеем. Итак, воспользуемся формулой (25), положив

$$r = n + k - 1, \quad m = 2k, \quad s = 0, \quad n = m - 1$$

В результате получим

$$\sum_k \sum_{0\le j\le n+k-1} \binom{n+k-1-j}{2k}\binom{2k}{k}\binom{j}{m-1}\frac{(-1)^k}{k+1}. \tag{29}$$

Мы хотим выполнить суммирование сначала по k, но для изменения порядка суммирования требуется, чтобы суммирование происходило по тем k, которые ≥ 0 и $\ge j-n+1$. К сожалению, последнее условие вызывает проблемы, так как при $j \ge n$ сумма *не* определена. Попытаемся спасти ситуацию. Для начала заметим, что члены суммы (29) равны нулю при $n \le j \le n+k-1$. Отсюда следует, что $k \ge 1$; таким образом, $0 \le n+k-1-j \le k-1 < 2k$ и первый биномиальный коэффициент в формуле (29) обращается в нуль. Следовательно, во второй сумме можно выполнять суммирование по $0 \le j < n$ и теперь с изменением порядка суммирования никаких проблем не будет. Суммируя по k согласно (28), получим

$$\sum_{0\le j<n} \binom{j}{m-1}\delta_{(n-1-j)0}.$$

Все члены этой суммы, за исключением того, который соответствует $j = n-1$, обращаются в нуль. Таким образом, окончательно получаем

$$\binom{n-1}{m-1}.$$

Эта задача оказалась довольно сложной, но все-таки вполне разрешимой; все наши действия были обоснованы. Советую вам внимательно изучить решение, так как в нем продемонстрированы тонкие моменты, связанные с ограничениями, которые наложены на тождества. Но на самом деле существует более оптимальный метод решения этой задачи; читателю предоставляется самостоятельно найти способ преобразования исходной суммы таким образом, чтобы к ней можно было применить формулу (26) (см. упр. 30).

Пример 4. Докажите, что

$$\sum_k A_k(r,t)A_{n-k}(s,t) = A_n(r+s,t), \quad \text{целое } n \ge 0, \tag{30}$$

где $A_n(x,t)$ — многочлен степени n по x, такой, что

$$A_n(x,t) = \binom{x-nt}{n}\frac{x}{x-nt} \quad \text{для } x \ne nt.$$

Решение. Можно предположить, что $r \neq kt \neq s$ для $0 \leq k \leq n$, так как обе части (30) — это многочлены по r, s, t. Вычислим сумму

$$\sum_k \binom{r-kt}{k} \binom{s-(n-k)t}{n-k} \frac{r}{r-kt} \frac{s}{s-(n-k)t},$$

которая кажется еще более страшной, чем во всех предыдущих задачах! Тем не менее обратите внимание на сильное сходство с формулой (26), а также на случай $t = 0$.

Возникает искушение заменить

$$\binom{r-kt}{k} \frac{r}{r-kt} \quad \text{выражением} \quad \binom{r-kt-1}{k-1} \frac{r}{k},$$

но только при этом последнее выражение теряет сходство с (26) и не имеет смысла при $k = 0$. Поэтому лучше всего воспользоваться методом *элементарных дробей*, при котором дробь со сложным знаменателем можно заменить суммой дробей с более простыми знаменателями. В результате имеем

$$\frac{1}{r-kt} \frac{1}{s-(n-k)t} = \frac{1}{r+s-nt} \left(\frac{1}{r-kt} + \frac{1}{s-(n-k)t} \right).$$

Подставляя это выражение в исходную сумму, получаем

$$\frac{s}{r+s-nt} \sum_k \binom{r-kt}{k} \binom{s-(n-k)t}{n-k} \frac{r}{r-kt} + \frac{r}{r+s-nt} \sum_k \binom{r-kt}{k} \binom{s-(n-k)t}{n-k} \frac{s}{s-(n-k)t}.$$

И теперь, если во второй сумме выполнить замену k на $n-k$, то по формуле (26) можно вычислить обе суммы; отсюда непосредственно получаем искомый результат. Тождества (26) и (30) получены Г. А. Роте (H. A. Rothe) и опубликованы в его книге *Formulæ de Serierum Reversione* (Leipzig, 1793), а частные случаи этих формул время от времени продолжают "открывать". Об интересной истории данных тождеств и некоторых их обобщениях речь идет в статье Г. В. Гоулд (H. W. Gould) и Дж. Кауцки (J. Kaucký), *Journal of Combinatorial Theory* **1** (1966), 233–248.

Пример 5. Как определить значения $a_0, a_1, a_2, \ldots$, такие, что

$$n! = a_0 + a_1 n + a_2 n(n-1) + a_3 n(n-1)(n-2) + \cdots \tag{31}$$

для всех неотрицательных целых n?

Решение. Ответ на этот вопрос дает равенство 1.2.5–(11), которое было приведено без доказательства в предыдущем разделе. Давайте сделаем вид, что ответ нам пока еще неизвестен. Очевидно, что задача имеет решение, так как можно положить $n = 0$ и определить a_0, затем положить $n = 1$ и определить a_1, и т. д.

Сначала запишем (31) с помощью биномиальных коэффициентов:

$$n! = \sum_k \binom{n}{k} k!\, a_k \tag{32}$$

Задача решения уравнений относительно a_k, подобных этому, называется *задачей обращения*. Метод, которым мы воспользуемся, применим для решения всех аналогичных задач.

Идея решения основана на частном случае тождества (23) для $s = 0$:

$$\sum_k \binom{r}{k}\binom{k}{n}(-1)^{r-k} = \binom{0}{n-r} = \delta_{nr}, \quad \text{целое } n, \text{целое } r \geq 0. \tag{33}$$

Данная формула важна потому, что при $n \neq r$ сумма равна нулю. Это позволяет без труда решить задачу, поскольку многие члены суммы оказываются равными нулю, как в примере 3:

$$\begin{aligned}\sum_n n!\binom{m}{n}(-1)^{m-n} &= \sum_n \sum_k \binom{n}{k} k!\, a_k \binom{m}{n}(-1)^{m-n} \\ &= \sum_k k!\, a_k \sum_n \binom{n}{k}\binom{m}{n}(-1)^{m-n} \\ &= \sum_k k!\, a_k \delta_{km} = m!\, a_m.\end{aligned}$$

Обратите внимание на то, как нам удалось получить уравнение, в котором содержится только один неизвестный коэффициент — a_m. Для этого мы сложили равенства (32) для $n = 0, 1, \ldots$, умноженные на подходящие коэффициенты. В результате имеем

$$a_m = \sum_{n\geq 0}(-1)^{m-n}\frac{n!}{m!}\binom{m}{n} = \sum_{0\leq n\leq m}\frac{(-1)^{m-n}}{(m-n)!} = \sum_{0\leq n\leq m}\frac{(-1)^n}{n!}.$$

Таким образом, задача 5 решена. А теперь давайте более подробно рассмотрим, что означает соотношение (33). Для неотрицательных целых r и m имеем

$$\sum_k \binom{r}{k}(-1)^{r-k}\left(c_0\binom{k}{0} + c_1\binom{k}{1} + \cdots + c_m\binom{k}{m}\right) = c_r,$$

так как остальные члены после суммирования исчезают. Выбирая соответствующим образом коэффициенты c_i, можно представить *любой* многочлен от k в виде суммы биномиальных коэффициентов с верхним индексом k. Поэтому

$$\sum_k \binom{r}{k}(-1)^{r-k}(b_0 + b_1 k + \cdots + b_r k^r) = r!\, b_r, \quad \text{целое } r \geq 0, \tag{34}$$

где $b_0 + \cdots + b_r k^r$ — произвольный многочлен степени не выше r. (Эта формула не должна быть большой неожиданностью для студентов, изучающих численный анализ, так как $\sum_k \binom{r}{k}(-1)^{r-k} f(x+k)$ — это "r-я разность" функции $f(x)$.)

Из (34) можно непосредственно получить многие другие соотношения, которые кажутся сложными на первый взгляд и часто сопровождаются очень длинными доказательствами, например

$$\sum_k \binom{r}{k}\binom{s-kt}{r}(-1)^k = t^r, \quad \text{целое } r \geq 0. \tag{35}$$

В учебниках, подобных нашему, обычно приводится множество впечатляющих примеров использования хитроумных приемов, но никогда не упоминаются простые с виду задачи, в которых эти приемы не работают. Приведенные выше примеры, возможно, создали у вас впечатление, что с биномиальными коэффициентами можно делать все, что угодно. Но необходимо заметить, что, несмотря на формулы (10), (11) и (18), похоже, не существует простой формулы для аналогичной суммы

$$\sum_{k=0}^{n}\binom{m}{k} = \binom{m}{0} + \binom{m}{1} + \cdots + \binom{m}{n}$$

при $n < m$. (При $n = m$ существует простая формула. Как она выглядит, вы узнаете из упр. 36.)

С другой стороны, эта сумма приобретает законченный вид функции от n, если m — отрицательное целое число, например

$$\sum_{k=0}^{n}\binom{-2}{k} = (-1)^n\left\lceil\frac{n+1}{2}\right\rceil. \tag{37}$$

Существует также простая формула

$$\sum_{k=0}^{n}\binom{m}{k}\left(k-\frac{m}{2}\right) = -\frac{m}{2}\binom{m-1}{n} \tag{38}$$

для суммы, хотя, казалось бы, эта формула должна быть более сложной.

Как определить, что пора прекратить работу над суммой, которая не поддается дальнейшему упрощению? К счастью, теперь существует простой способ получения ответа на этот вопрос во многих важных случаях. Алгоритм Р. В. Госпера (R. W. Gosper) и Д. Зейльбергера (D. Zeilberger) позволяет получить замкнутые формы, если они существуют, выраженные через биномиальные коэффициенты, и доказать, что таких форм нет, если они не существуют. Рассмотрение алгоритма Госпера-Зейдельберга выходит за рамки этой книги, но о нем можно прочитать в журнале *CMath* §5.8. (См. также книгу Петковшека (Petkovšek), Вильфа (Wilf) и Зейльбергера (Zeilberger) $A = B$ (Wellesley, Mass.: A. K. Peters, 1996).)

Главным инструментом выполнения преобразований над суммами биномиальных коэффициентов систематическим и механическим путем является использование свойств *гипергеометрических функций*, которые представляют собой бесконечные суммы, определенные через возрастающие факториальные степени следующим образом:

$$F\left(\begin{matrix}a_1, \ldots, a_m\\ b_1, \ldots, b_n\end{matrix}\middle| z\right) = \sum_{k\ge 0}\frac{a_1^{\bar{k}}\ldots a_m^{\bar{k}}}{b_1^{\bar{k}}\ldots b_n^{\bar{k}}}\frac{z^k}{k!}. \tag{39}$$

Вводная информация об этих важных функциях содержится в разделах 5.5 и 5.6 *CMath*. Исторические сведения о данных функциях приводятся также в книге J. Dutka, *Archive for History of Exact Sciences* **31** (1984), 15–34.

Существует несколько важных обобщений понятия биномиальных коэффициентов, о которых мы сейчас вкратце поговорим. Во-первых, в качестве нижнего индекса k в $\binom{r}{k}$ можно рассмотреть произвольные действительные числа; см. упр. 40–45.

Во-вторых, существует также обобщенная формула

$$\binom{r}{k}_q = \frac{(1-q^r)(1-q^{r-1})\dots(1-q^{r-k+1})}{(1-q^k)(1-q^{k-1})\dots(1-q^1)}, \tag{40}$$

которая принимает вид обычного биномиального коэффициента $\binom{r}{k}_1 = \binom{r}{k}$, если в (40) перейти к пределу при q, стремящемся к 1. Это станет очевидным, если разделить каждый сомножитель числителя и знаменателя на $1-q$. Основные свойства таких "q-номиальных коэффициентов" обсуждаются в упр. 58.

Но для наших целей самым важным обобщением является ***полиномиальный коэффициент***

$$\binom{k_1+k_2+\cdots+k_m}{k_1, k_2, \dots, k_m} = \frac{(k_1+k_2+\cdots+k_m)!}{k_1!\,k_2!\dots k_m!}, \quad \text{целое } k_i \ge 0. \tag{41}$$

Главное свойство полиномиальных коэффициентов выражается обобщением соотношения (13):

$$(x_1+x_2+\cdots+x_m)^n = \sum_{k_1+k_2+\cdots+k_m=n} \binom{n}{k_1, k_2, \dots, k_m} x_1^{k_1} x_2^{k_2} \dots x_m^{k_m}. \tag{42}$$

Важно отметить, что любой полиномиальный коэффициент можно выразить через биномиальные коэффициенты

$$\binom{k_1+k_2+\cdots+k_m}{k_1, k_2, \dots, k_m} = \binom{k_1+k_2}{k_1}\binom{k_1+k_2+k_3}{k_1+k_2}\cdots\binom{k_1+k_2+\cdots+k_m}{k_1+\cdots+k_{m-1}} \tag{43}$$

и, следовательно, применить уже известные нам методы работы с биномиальными коэффициентами. В обеих частях тождества (20) содержится триномиальный коэффициент

$$\binom{r}{k,\, m-k,\, r-m}.$$

И в завершение этого раздела дадим краткий анализ преобразования многочлена, представленного в виде степеней переменной x, в многочлен, выраженный через биномиальные коэффициенты. Коэффициенты, фигурирующие в таком преобразовании, называются ***числами Стирлинга***; эти числа возникают при изучении самых разнообразных алгоритмов.

Числа Стирлинга бывают двух видов. Числа Стирлинга первого рода обозначим через $\left[{n \atop k}\right]$, а второго рода — через $\left\{{n \atop k}\right\}$. Эти обозначения, введенные Йованом Карамата (Jovan Karamata) [*Mathematica* (Cluj) **9** (1935), 164–178], имеют неоспоримые преимущества над многими другими [см. D. E. Knuth, *AMM* **99** (1992), 403–422]. Фигурные скобки в записи $\left\{{n \atop k}\right\}$ легко запомнить потому, что они обычно обозначают множество, а $\left\{{n \atop k}\right\}$ — это число способов разбиения множества из n элементов на k непересекающихся подмножеств (упр. 64). Числа Стирлинга первого рода $\left[{n \atop k}\right]$ также имеют комбинаторную интерпретацию, которая будет подробно рассмотрена в разделе 1.3.3: $\left[{n \atop k}\right]$ — это число перестановок n букв с k циклами.

В табл. 2 представлены треугольники Стирлинга, в некоторых отношениях аналогичные треугольнику Паскаля.

Таблица 2

ЧИСЛА СТИРЛИНГА ПЕРВОГО И ВТОРОГО РОДА

n	$\begin{bmatrix} n \\ 0 \end{bmatrix}$	$\begin{bmatrix} n \\ 1 \end{bmatrix}$	$\begin{bmatrix} n \\ 2 \end{bmatrix}$	$\begin{bmatrix} n \\ 3 \end{bmatrix}$	$\begin{bmatrix} n \\ 4 \end{bmatrix}$	$\begin{bmatrix} n \\ 5 \end{bmatrix}$	$\begin{bmatrix} n \\ 6 \end{bmatrix}$	$\begin{bmatrix} n \\ 7 \end{bmatrix}$	$\begin{bmatrix} n \\ 8 \end{bmatrix}$
0	1	0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0	0
2	0	1	1	0	0	0	0	0	0
3	0	2	3	1	0	0	0	0	0
4	0	6	11	6	1	0	0	0	0
5	0	24	50	35	10	1	0	0	0
6	0	120	274	225	85	15	1	0	0
7	0	720	1764	1624	735	175	21	1	0
8	0	5040	13068	13132	6769	1960	322	28	1

n	$\begin{Bmatrix} n \\ 0 \end{Bmatrix}$	$\begin{Bmatrix} n \\ 1 \end{Bmatrix}$	$\begin{Bmatrix} n \\ 2 \end{Bmatrix}$	$\begin{Bmatrix} n \\ 3 \end{Bmatrix}$	$\begin{Bmatrix} n \\ 4 \end{Bmatrix}$	$\begin{Bmatrix} n \\ 5 \end{Bmatrix}$	$\begin{Bmatrix} n \\ 6 \end{Bmatrix}$	$\begin{Bmatrix} n \\ 7 \end{Bmatrix}$	$\begin{Bmatrix} n \\ 8 \end{Bmatrix}$
0	1	0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0	0
2	0	1	1	0	0	0	0	0	0
3	0	1	3	1	0	0	0	0	0
4	0	1	7	6	1	0	0	0	0
5	0	1	15	25	10	1	0	0	0
6	0	1	31	90	65	15	1	0	0
7	0	1	63	301	350	140	21	1	0
8	0	1	127	966	1701	1050	266	28	1

Аппроксимация при больших n приведена в работе L. Moser, M. Wyman, *J. London Math. Soc.* **33** (1958), 133–146; *Duke Math. J.* **25** (1958), 29–43; D. E. Barton, F. N. David, M. Merrington, *Biometrika* **47** (1960), 439–445; **50** (1963), 169–176; N. M. Temme, *Studies in Applied Math.* **89** (1993), 233–243; H. S. Wilf, *J. Combinatorial Theory* **A64** (1993), 344–349; H.-K. Hwang, *J. Combinatorial Theory* **A71** (1995), 343–351.

Числа Стирлинга первого рода используются для перехода от факториальных степеней к обычным:

$$\begin{aligned} x^{\underline{n}} &= x(x-1)\dots(x-n+1) \\ &= \begin{bmatrix} n \\ n \end{bmatrix} x^n - \begin{bmatrix} n \\ n-1 \end{bmatrix} x^{n-1} + \dots + (-1)^n \begin{bmatrix} n \\ 0 \end{bmatrix} \\ &= \sum_k (-1)^{n-k} \begin{bmatrix} n \\ k \end{bmatrix} x^k. \end{aligned} \tag{44}$$

Например, согласно табл. 2 имеем

$$\binom{x}{5} = \frac{x^{\underline{5}}}{5!} = \frac{1}{120}(x^5 - 10x^4 + 35x^3 - 50x^2 + 24x).$$

Числа Стирлинга второго рода используются для перехода от обычных степеней к факториальным:

$$x^n = \left\{ {n \atop n} \right\} x^{\underline{n}} + \cdots + \left\{ {n \atop 1} \right\} x^{\underline{1}} + \left\{ {n \atop 0} \right\} x^{\underline{0}} = \sum_k \left\{ {n \atop k} \right\} x^{\underline{k}}. \tag{45}$$

Фактически именно эта формула послужила причиной того, что Стирлинг занялся изучением чисел $\left\{ {n \atop k} \right\}$ в работе *Methodus Differentialis* (London, 1730). Например, из табл. 2 имеем

$$\begin{aligned} x^5 &= x^{\underline{5}} + 10x^{\underline{4}} + 25x^{\underline{3}} + 15x^{\underline{2}} + x^{\underline{1}} \\ &= 120\binom{x}{5} + 240\binom{x}{4} + 150\binom{x}{3} + 30\binom{x}{2} + \binom{x}{1}. \end{aligned}$$

А теперь приведем наиболее важные тождества, в которых фигурируют числа Стирлинга. В этих тождествах переменные m и n всегда обозначают неотрицательные целые числа.

Формулы сложения:

$$\begin{aligned} \left[{n+1 \atop m} \right] &= n\left[{n \atop m} \right] + \left[{n \atop m-1} \right]; \\ \left\{ {n+1 \atop m} \right\} &= m\left\{ {n \atop m} \right\} + \left\{ {n \atop m-1} \right\}. \end{aligned} \tag{46}$$

Формулы обращения (ср. с (33)):

$$\sum_k \left[{n \atop k} \right] \left\{ {k \atop m} \right\} (-1)^{n-k} = \delta_{mn}, \quad \sum_k \left\{ {n \atop k} \right\} \left[{k \atop m} \right] (-1)^{n-k} = \delta_{mn}. \tag{47}$$

Некоторые частные значения:

$$\binom{0}{n} = \left[{0 \atop n} \right] = \left\{ {0 \atop n} \right\} = \delta_{n0}, \qquad \binom{n}{n} = \left[{n \atop n} \right] = \left\{ {n \atop n} \right\} = 1; \tag{48}$$

$$\left[{n \atop n-1} \right] = \left\{ {n \atop n-1} \right\} = \binom{n}{2}; \tag{49}$$

$$\left[{n+1 \atop 0} \right] = \left\{ {n+1 \atop 0} \right\} = 0, \quad \left[{n+1 \atop 1} \right] = n!, \quad \left\{ {n+1 \atop 1} \right\} = 1, \quad \left\{ {n+1 \atop 2} \right\} = 2^n - 1. \tag{50}$$

Формулы разложения:

$$\sum_k \left[{n \atop k} \right] \binom{k}{m} = \left[{n+1 \atop m+1} \right], \quad \sum_k \left[{n+1 \atop k+1} \right] \binom{k}{m} (-1)^{k-m} = \left[{n \atop m} \right]; \tag{51}$$

$$\sum_k \left\{ {k \atop m} \right\} \binom{n}{k} = \left\{ {n+1 \atop m+1} \right\}, \quad \sum_k \left\{ {k+1 \atop m+1} \right\} \binom{n}{k} (-1)^{n-k} = \left\{ {n \atop m} \right\}; \tag{52}$$

$$\sum_k \binom{m}{k} (-1)^{m-k} k^n = m! \left\{ {n \atop m} \right\}; \tag{53}$$

$$\sum_k \binom{m-n}{m+k}\binom{m+n}{n+k}\left\{{m+k \atop k}\right\} = \left[{n \atop n-m}\right],$$
$$\sum_k \binom{m-n}{m+k}\binom{m+n}{n+k}\left[{m+k \atop k}\right] = \left\{{n \atop n-m}\right\}; \tag{54}$$

$$\sum_k \left\{{n+1 \atop k+1}\right\}\left[{k \atop m}\right](-1)^{k-m} = \binom{n}{m}; \tag{55}$$

$$\sum_{k\le n}\left[{k \atop m}\right]\frac{n!}{k!} = \left[{n+1 \atop m+1}\right], \qquad \sum_{k\le n}\left\{{k \atop m}\right\}(m+1)^{n-k} = \left\{{n+1 \atop m+1}\right\}. \tag{56}$$

Другие фундаментальные тождества, связанные с числами Стирлинга, приведены в упр. 1.2.6-61 и 1.2.7-6 и в разделе 1.2.9 (соотношения (23) и (26)–(28)).

Тождество (49) — это только один пример общей закономерности: числа Стирлинга $\left[{n \atop n-m}\right]$ и $\left\{{n \atop n-m}\right\}$ являются многочленами от n степени $2m$, где m — неотрицательное целое число. Например, для $m = 2$ и $m = 3$ получим следующие формулы:

$$\left[{n \atop n-2}\right] = \binom{n}{4} + 2\binom{n+1}{4}, \qquad \left\{{n \atop n-2}\right\} = \binom{n+1}{4} + 2\binom{n}{4},$$
$$\left[{n \atop n-3}\right] = \binom{n}{6} + 8\binom{n+1}{6} + 6\binom{n+2}{6}; \quad \left\{{n \atop n-3}\right\} = \binom{n+2}{6} + 8\binom{n+1}{6} + 6\binom{n}{6}. \tag{57}$$

Поэтому имеет смысл определить числа $\left[{r \atop r-m}\right]$ и $\left\{{r \atop r-m}\right\}$ для произвольных действительных (или комплексных) значений r. Используя это обобщение, получаем следующую интересную связь между числами Стирлинга двух родов:

$$\left\{{n \atop m}\right\} = \left[{-m \atop -n}\right] \tag{58}$$

Такая связь называется законом двойственности, который содержался в неявной форме в оригинальных выкладках Стирлинга. Более того, соотношение (45), в целом, остается справедливым в том смысле, что бесконечные ряды

$$z^r = \sum_k \left\{{r \atop r-k}\right\} z^{\underline{r-k}} \tag{59}$$

сходятся, когда действительная часть z положительна. Вторая формула, т. е. (44) аналогичным образом обобщается на случай асимптотических (но не сходящихся) рядов:

$$z^r = \sum_{k=0}^{m}\left[{r \atop r-k}\right](-1)^k z^{r-k} + O(z^{r-m-1}). \tag{60}$$

(См. упр. 65.) В разделах 6.1, 6.2 и 6.5 *CMath* содержится дополнительная информация о числах Стирлинга и о том, как оперировать ими в формулах. См. также упр. 4.7-21, в котором будет рассмотрено общее семейство треугольников, включающее числа Стирлинга в качестве частного случая.

УПРАЖНЕНИЯ

1. [*00*] Сколько существует сочетаний из n по $n-1$?

2. [*00*] Чему равно $\binom{0}{0}$?

3. [*00*] Сколько существует различных способов сдать карты одному игроку во время игры в бридж (13 карт из колоды, состоящей из 52 карт)?

4. [*10*] Представьте ответ к задаче 3 в виде произведения простых чисел.

▶ **5.** [*05*] С помощью треугольника Паскаля объясните тот факт, что $11^4 = 14641$.

▶ **6.** [*10*] Треугольник Паскаля (см. табл. 1) можно продолжить во всех направлениях с помощью формулы сложения (9). Добавьте к табл. 1 три строки *сверху* (т. е. для $r = -1$, -2 и -3).

7. [*12*] Если n — фиксированное положительное целое, то при каком значении k $\binom{n}{k}$ принимает максимальное значение?

8. [*00*] Какое свойство треугольника Паскаля отражено в "свойстве симметрии" (6)?

9. [*01*] Чему равно $\binom{n}{n}$? (Рассмотрите все целые n.)

▶ **10.** [*M25*] Пусть p — простое число. Покажите следующее.

a) $\binom{n}{p} \equiv \left\lfloor \frac{n}{p} \right\rfloor$ (по модулю p).

b) $\binom{p}{k} \equiv 0$ (по модулю p) для $1 \le k \le p-1$.

c) $\binom{p-1}{k} \equiv (-1)^k$ (по модулю p) для $0 \le k \le p-1$.

d) $\binom{p+1}{k} \equiv 0$ (по модулю p) для $2 \le k \le p-1$.

e) (Э. Люка (É. Lucas), 1877.)

$$\binom{n}{k} \equiv \binom{\lfloor n/p \rfloor}{\lfloor k/p \rfloor} \binom{n \bmod p}{k \bmod p} \quad \text{(по модулю } p\text{)}.$$

f) Если в системе счисления с основанием p числа n и k представляются в виде

$$\begin{aligned} n &= a_r p^r + \cdots + a_1 p + a_0, \\ k &= b_r p^r + \cdots + b_1 p + b_0, \end{aligned} \quad \text{то} \quad \binom{n}{k} \equiv \binom{a_r}{b_r} \cdots \binom{a_1}{b_1} \binom{a_0}{b_0} \quad \text{(по модулю } p\text{)}.$$

▶ **11.** [*M20*] (Э. Куммер (E. Kummer), 1852.) Пусть p — простое число. Покажите, что если p^n делит

$$\binom{a+b}{a},$$

а p^{n+1} — нет, то n равно числу *переносов*, которые выполняются при сложении чисел a и b в системе счисления с основанием p. [*Указание.* См. упр. 1.2.5–12.]

12. [*M22*] Существуют ли целые положительные числа n, для которых все ненулевые элементы в n-й строке треугольника Паскаля являются *нечетными*? Если да, найдите все такие n.

13. [*M13*] Докажите формулу суммирования (10).

14. [*M21*] Вычислите сумму $\sum_{k=0}^{n} k^4$.

15. [*M15*] Докажите биномиальную формулу (13).

16. [*M15*] Для положительных целых n и k докажите свойство симметрии

$$(-1)^n\binom{-n}{k-1} = (-1)^k\binom{-k}{n-1}$$

▸ **17.** [*M18*] На основании соотношения (15) и тождества $(1+x)^{r+s} = (1+x)^r(1+x)^s$ докажите формулу Чжу-Вандермонда (21).

18. [*M15*] На основании соотношений (21) и (6) докажите (22).

19. [*M18*] Докажите (23) по индукции.

20. [*M20*] Докажите (24) с помощью (21) и (19), а затем покажите, что еще одно применение формулы (19) дает (25).

▸ **21.** [*M05*] Обе части равенства (25) — это многочлены по s; почему это равенство не является тождеством по s?

22. [*M20*] Докажите (26) для частного случая $s = n-1-r+nt$.

23. [*M13*] Предполагая, что (26) выполняется для (r, s, t, n) и $(r,\ s-t,\ t,\ n-1)$, докажите его для $(r,\ s+1,\ t,\ n)$.

24. [*M15*] Объясните, как объединить результаты двух предыдущих упражнений для доказательства (26).

25. [*HM30*] Пусть многочлен $A_n(x,t)$ определяется формулой (30). Пусть $z = x^{t+1} - x^t$. Докажите, что $\sum_k A_k(r,t)z^k = x^r$, если z достаточно мало. [*Замечание.* При $t = 0$ этот результат, по существу, совпадает с биномиальной теоремой, поэтому приведенное соотношение является важным обобщением биномиальной теоремы. При доказательстве тождества биномиальную теорему можно считать известной.] *Указание.* Начните с тождества

$$\sum_j (-1)^j\binom{k}{j}\binom{r-jt}{k}\frac{r}{r-jt} = \delta_{k0}.$$

26. [*HM25*] Используя предположения из предыдущего упражнения, докажите, что

$$\sum_k \binom{r-tk}{k} z^k = \frac{x^{r+1}}{(t+1)x-t}.$$

27. [*HM21*] Решите задачу из примера 4, приведенного в тексте раздела, используя результат упр. 25, и на основании двух предыдущих упражнений докажите (26). [*Указание.* См. упр. 17.]

28. [*M25*] Докажите, что

$$\sum_k \binom{r+tk}{k}\binom{s-tk}{n-k} = \sum_{k\geq 0}\binom{r+s-k}{n-k}t^k,$$

если n — неотрицательное целое число.

29. [*M20*] Покажите, что тождество (34) — это просто частный случай общего тождества, доказанного в упр. 1.2.3–33.

▸ **30.** [*M24*] Покажите, что существует более удачный (по сравнению с приведенным в тексте раздела) способ решения примера 3, который состоит в преобразовании суммы с применением соотношения (26).

▸ **31.** [*M20*] Выразите сумму

$$\sum_k \binom{m-r+s}{k}\binom{n+r-s}{n-k}\binom{r+k}{m+n}$$

через r, s, m и n, если m и n — неотрицательные целые числа. Начните с замены

$$\binom{r+k}{m+n}^{k} \quad \text{суммой} \quad \sum_{j}\binom{r}{m+n-j}\binom{k}{j}.$$

32. [*M20*] Покажите, что $\sum_k \left[{n \atop k}\right] x^k = x^{\overline{n}}$, где $x^{\overline{n}}$ — возрастающая факториальная степень, определенная формулой 1.2 5–(19).

33. [*M20*] (*Сумма Капелли*) Покажите, что биномиальная формула справедлива и в том случае, когда в ней вместо обычных степеней фигурируют возрастающие факториальные степени, т. е. докажите тождество $(x+y)^{\overline{n}} = \sum_k \binom{n}{k} x^{\overline{k}} y^{\overline{n-k}}$.

34. [*M23*] (*Сумма Торелли*) В свете предыдущего упражнения покажите, что обобщение Абеля (16) для биномиальной формулы справедливо также для возрастающих степеней:

$$(x+y)^{\overline{n}} = \sum_k \binom{n}{k} x(x-kz+1)^{\overline{k-1}}(y+kz)^{\overline{n-k}}.$$

35. [*M23*] Выведите формулы сложения (46) для чисел Стирлинга непосредственно из определений (44) и (45).

36. [*M10*] Чему равна сумма $\sum_k \binom{n}{k}$ чисел каждой строки треугольника Паскаля? Чему равна сумма этих чисел, взятых с чередующимися знаками, $\sum_k \binom{n}{k}(-1)^k$?

37. [*M10*] Используя результаты предыдущего упражнения, вычислите сумму элементов строки, взятых через один: $\binom{n}{0} + \binom{n}{2} + \binom{n}{4} + \cdots$.

38. [*HM30*] (К Рамус (C Ramus), 1834) Обобщая результат предыдущего упражнения, покажите, что для $0 \le k < m$ справедлива следующая формула

$$\binom{n}{k} + \binom{n}{m+k} + \binom{n}{2m+k} + \cdots = \frac{1}{m} \sum_{0 \le j < m} \left(2\cos\frac{j\pi}{m}\right)^n \cos\frac{j(n-2k)\pi}{m}.$$

Например,

$$\binom{n}{1} + \binom{n}{4} + \binom{n}{7} + \cdots = \frac{1}{3}\left(2^n + 2\cos\frac{(n-2)\pi}{3}\right).$$

[*Указание.* Найдите нужную линейную комбинацию этих коэффициентов, умноженных на корни m-й степени из единицы.] Данное тождество особенно замечательно при $m \ge n$

39. [*M10*] Чему равна сумма $\sum_k \left[{n \atop k}\right]$ чисел каждой строки первого треугольника Стирлинга? Чему равна сумма этих чисел, взятых с чередующимися знаками? (См. упр. 36.)

40. [*HM17*] Для положительных действительных чисел x, y *бета-функция* $\mathrm{B}(x,y)$ определяется формулой $\mathrm{B}(x,y) = \int_0^1 t^{x-1}(1-t)^{y-1}\,dt$.

a) Покажите, что $\mathrm{B}(x,1) = \mathrm{B}(1,x) = 1/x$.

b) Покажите, что $\mathrm{B}(x+1,\,y) + \mathrm{B}(x,\,y+1) = \mathrm{B}(x,y)$

c) Покажите, что $\mathrm{B}(x,y) = ((x+y)/y)\,\mathrm{B}(x,\,y+1)$.

41. [*HM22*] В упр. 1 2.5–19 мы установили связь между гамма-функцией и бета-функцией, показав, что $\Gamma_m(x) = m^x \mathrm{B}(x, m+1)$, если m — положительное целое.

a) Докажите, что

$$\mathrm{B}(x,y) = \frac{\Gamma_m(y)m^x}{\Gamma_m(x+y)}\mathrm{B}(x,\,y+m+1).$$

b) Покажите, что

$$\mathrm{B}(x,y) = \frac{\Gamma(x)\Gamma(y)}{\Gamma(x+y)}$$

42. [*HM10*] Выразите биномиальный коэффициент $\binom{r}{k}$ через бета-функцию, определенную выше (Это позволит распространить определение биномиальных коэффициентов на все действительные значения k)

43. [*HM20*] Покажите, что $B(1/2,1/2)=\pi$ (Тогда на основании упр 41 можно заключить, что $\Gamma(1/2)=\sqrt{\pi}$)

44. [*HM20*] Используя обобщение биномиальных коэффициентов, предложенное в упр 42, покажите, что

$$\binom{r}{1/2}=2^{2r+1}\Big/\binom{2r}{r}\pi$$

45. [*HM21*] Используя обобщение биномиальных коэффициентов, предложенное в упр 42, найдите $\lim_{r\to\infty}\binom{r}{k}/r^k$

▸ **46.** [*M21*] Используя формулу Стирлинга и соотношение 1 2 5–(7), найдите приближенное значение $\binom{x+y}{y}$ для больших x и y В частности, найдите приближенное значение $\binom{2n}{n}$ для больших n

47. [*M21*] Покажите, что для целых k

$$\binom{r}{k}\binom{r-1/2}{k}=\binom{2r}{k}\binom{2r-k}{k}\Big/4^k=\binom{2r}{2k}\binom{2k}{k}\Big/4^k$$

Приведите более простую формулу для частного случая $r=-1/2$

▸ **48.** [*M25*] Покажите, что

$$\sum_{k\ge 0}\binom{n}{k}\frac{(-1)^k}{k+x}=\frac{\boldsymbol{n!}}{x(x+1)\quad(x+n)}=\frac{1}{x\binom{n+x}{n}},$$

если знаменатели не обращаются в нуль [Обратите внимание, что эта формула дает обратное значение биномиального коэффициента а также разложение $1/x(x+1)\quad(x+n)$ на элементарные дроби]

49. [*M20*] Покажите, что из тождества $(1+x)^r=(1-x^2)^r(1-x)^{-r}$ следует соотношение для биномиальных коэффициентов

50. [*M20*] Докажите формулу Абеля (16) для частного случая $x+y=0$

51. [*M21*] Докажите формулу Абеля (16) следующим способом запишите y в виде $y=(x+y)-x$, разложите правую часть по степеням $(x+y)$ и примените результат предыдущего упражнения

52. [*HM11*] Докажите, что биномиальная формула Абеля (16) не всегда справедлива, если n не является неотрицательным целым числом Для этого вычислите значение правой части при $n=x=-1$, $y=z=1$

53. [*M25*] (a) Докажите следующее тождество индукцией по m, если m и n — целые

$$\sum_{k=0}^{m}\binom{r}{k}\binom{s}{n-k}(nr-(r+s)k)=(m+1)(n-m)\binom{r}{m+1}\binom{s}{n-m}$$

(b) Используя важные соотношения из упр 47,

$$\binom{-1/2}{n}=\frac{(-1)^n}{2^{2n}}\binom{2n}{n},\quad\binom{1/2}{n}=\frac{(-1)^{n-1}}{2^{2n}(2n-1)}\binom{2n}{n}=\frac{(-1)^{n-1}}{2^{2n-1}(2n-1)}\binom{2n-1}{n}-\delta_{n0},$$

покажите, что следующую формулу можно получить как частный случай тождества из п (a)

онжоь $$\sum_{k=0}^{m}\binom{2k-1}{k}\binom{2n-2k}{n-k}\frac{-1}{2k-1}=\frac{n-m}{2n}\binom{2m}{m}\binom{2n-2m}{n-m}+\frac{1}{2}\binom{2n}{n}$$

(Это значительно более общий результат, чем соотношение (26) для случая $r = -1$, $s = 0$, $t = -2$.)

54. [*M21*] Рассмотрите треугольник Паскаля (см. табл 1) как матрицу Найдите *обратную* матрицу.

55. [*M21*] Рассматривая каждый треугольник Стирлинга (см. табл. 2) в качестве матрицы, найдите обратные к ним матрицы.

56. [*20*] (*Комбинаторная числовая система.*) Для каждого целого $n = 0, 1, 2, \ldots, 20$ найдите три целых a, b, c, таких, что $n = \binom{a}{1} + \binom{b}{2} + \binom{c}{3}$ и $0 \le a < b < c$. Как можно продолжить эту последовательность для бóльших значений n?

▸ **57.** [*M22*] Покажите, что коэффициент a_m в формуле Стирлинга 1.2.5–(12), в которой он пытался обобщить факториальную функцию, равен

$$\frac{(-1)^m}{m!}\sum_{k\ge 1}(-1)^k\binom{m-1}{k-1}\ln k.$$

58. [*M23*] Используя обозначения соотношения (40), докажите "q-номиальную теорему":

$$(1+x)(1+qx)\ldots\ (1+q^{n-1}x) = \sum_k \binom{n}{k}_q q^{k(k-1)/2}x^k.$$

Найдите q-номиальные обобщения фундаментальных тождеств (17) и (21).

59. [*M25*] Последовательность чисел A_{nk}, $n \ge 0$, $k \ge 0$, удовлетворяет соотношениям $A_{n0} = 1$, $A_{0k} = \delta_{0k}$, $A_{nk} = A_{(n-1)k} + A_{(n-1)(k-1)} + \binom{n}{k}$ для $nk > 0$. Найдите A_{nk}.

▸ **60.** [*M23*] Как вы уже знаете, $\binom{n}{k}$ — это число сочетаний из n элементов по k, т. е. число способов выбора k различных элементов из n-элементного множества. *Сочетания с повторениями* отличаются от обычных сочетаний только тем, что один элемент можно выбрать произвольное число раз. Таким образом, для сочетаний с повторениями список (1) следует продолжить, чтобы включить в него aaa, aab, aac, aad, aae, abb и т. д. Итак, сколько существует сочетаний с повторениями из n объектов по k?

61. [*M25*] Вычислите сумму

$$\sum_k \begin{bmatrix} n+1 \\ k+1 \end{bmatrix} \begin{Bmatrix} k \\ m \end{Bmatrix} (-1)^{k-m},$$

получив тем самым формулу, парную для (55).

▸ **62.** [*M23*] В тексте приводятся формулы для сумм, содержащих произведение двух биномиальных коэффициентов. Для сумм, содержащих произведение трех биномиальных коэффициентов, наиболее полезными будут тождество из упр. 31 и следующая формула:

$$\sum_k (-1)^k\binom{l+m}{l+k}\binom{m+n}{m+k}\binom{n+l}{n+k} = \frac{(l+m+n)!}{l!\,m!\,n!}, \quad \text{целое } l,m,n \ge 0.$$

(k пробегает как положительные, так и отрицательные значения.) Докажите это тождество. [*Указание.* Существует очень короткое доказательство, которое начинается с применения упр. 31.]

63. [*M30*] Если l, m и n — целые и $n \ge 0$, докажите, что

$$\sum_{j,k}(-1)^{j+k}\binom{j+k}{k+l}\binom{r}{j}\binom{n}{k}\binom{s+n-j-k}{m-j} = (-1)^l\binom{n+r}{n+l}\binom{s-r}{m-n-l}.$$

▸ **64.** [*M20*] Покажите, что $\left\{ {n \atop m} \right\}$ — это число способов разбиения множества из n элементов на m непустых непересекающихся подмножеств. Например, множество $\{1,2,3,4\}$ можно

разбить на два подмножества $\binom{4}{2} = 7$ способами. $\{1,2,3\}\{4\}$; $\{1,2,4\}\{3\}$; $\{1,3,4\}\{2\}$; $\{2,3,4\}\{1\}$; $\{1,2\}\{3,4\}$; $\{1,3\}\{2,4\}$; $\{1,4\}\{2,3\}$. *Указание.* Используйте соотношения (46).

65. [*HM35*] (Б. Ф. Логан (B. F. Logan).) Докажите формулы (59) и (60).

66. [*M29*] Пусть n — положительное целое число, а x и y — действительные числа, удовлетворяющие неравенству $n \le y \le x \le y+1$. Тогда $\binom{y}{n+1} \le \binom{x}{n+1} \le \binom{y+1}{n+1} = \binom{y}{n+1} + \binom{y}{n}$, так что существует единственное действительное число z, такое, что

$$\binom{x}{n+1} = \binom{y}{n+1} + \binom{z}{n}, \qquad n-1 \le z \le y.$$

Докажите, что

$$\binom{x}{n} \le \binom{y}{n} + \binom{z}{n-1}$$

[*Указание.* Рассмотрите представление $\binom{x}{n+1} = \binom{z+1}{n+1} + \sum_{k\ge 0}\binom{z-k}{n-k}\binom{x-z-1+k}{k+1}$.]

▸ **67.** [*M20*] Часто возникает необходимость в получении оценок для биномиальных коэффициентов. Докажите следующее неравенство, представляющее оценку сверху (заметим, что ее легко запомнить):

$$\binom{n}{k} \le \left(\frac{ne}{k}\right)^k, \qquad \text{где } n \ge k \ge 0.$$

68. [*M25*] (А. де Муавр (A. de Moivre).) Докажите, что для целого неотрицательного n

$$\sum_k \binom{n}{k} p^k (1-p)^{n-k} |k - np| = 2\lceil np\rceil \binom{n}{\lceil np\rceil} p^{\lceil np\rceil}(1-p)^{n+1-\lceil np\rceil}.$$

1.2.7. Гармонические числа

В дальнейшем для нас будет иметь большое значение следующая сумма:

$$H_n = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n} = \sum_{k=1}^{n} \frac{1}{k}, \quad n \ge 0. \tag{1}$$

Она не очень часто встречается в классической математике, и для нее не существует стандартного обозначения. Но в анализе алгоритмов она возникает почти на каждом шагу, поэтому будем использовать для нее обозначение H_n. (Помимо H_n, в математической литературе для этой суммы используются также обозначения h_n, S_n и $\psi(n+1)+\gamma$. Буква H обозначает "harmonic" ("гармонический"); H_n будем называть *гармоническими числами*, так как (1) обычно называют гармоническим рядом.)

На первый взгляд может показаться, что при больших n значение суммы H_n не слишком велико, так как мы постоянно добавляем все меньшие и меньшие числа. Но на самом деле можно показать, что H_n может достигать сколь угодно больших значений, если взять достаточно большое n, поскольку

$$H_{2^m} \ge 1 + \frac{m}{2}. \tag{2}$$

Эту оценку снизу можно получить, если заметить, что для $m \ge 0$

$$\begin{aligned} H_{2^{m+1}} &= H_{2^m} + \frac{1}{2^m+1} + \frac{1}{2^m+2} + \cdots + \frac{1}{2^{m+1}} \\ &\ge H_{2^m} + \frac{1}{2^{m+1}} + \frac{1}{2^{m+1}} + \cdots + \frac{1}{2^{m+1}} = H_{2^m} + \tfrac{1}{2}. \end{aligned}$$

Поэтому, когда m увеличивается на 1, левая часть неравенства (2) увеличивается по меньшей мере на $\frac{1}{2}$.

Но мы нуждаемся в более подробной информации о H_n, чем та, которую дает неравенство (2). Приближенная оценка H_n хорошо известна (по крайней мере, в математических кругах); она дается следующей формулой:

$$H_n = \ln n + \gamma + \frac{1}{2n} - \frac{1}{12n^2} + \frac{1}{120n^4} - \epsilon, \quad 0 < \epsilon < \frac{1}{252n^6}. \tag{3}$$

Здесь $\gamma = 0.5772156649\ldots$ — это *постоянная Эйлера*, введенная Леонардом Эйлером в работе *Commentarii Acad. Sci. Imp. Pet.* **7** (1734), 150–161. Точные значения H_n для малых n, а также значение γ с точностью до 40-го десятичного знака, приведены в приложении А. Формула (3) будет выведена в разделе 1.2.11.2.

Таким образом, H_n является достаточно близким к натуральному логарифму n. В упр. 7, (а) будет показано, что H_n и ведет себя в некоторой степени, как логарифмическая функция.

В этом смысле по мере увеличения n функция H_n стремится к бесконечности очень медленно, так как сумма

$$1 + \frac{1}{2^r} + \frac{1}{3^r} + \cdots + \frac{1}{n^r} \tag{4}$$

остается ограниченной для всех n, если показатель r — это любое действительное число, которое *больше* единицы (см. упр. 3). Сумму (4) обозначим через $H_n^{(r)}$.

Если показатель r в (4) больше или равен 2, то величина $H_n^{(r)}$ довольно близка к своему максимальному значению $H_\infty^{(r)}$ для всех n, кроме совсем малых. Величина $H_\infty^{(r)}$ хорошо известна в математике как *дзета-функция Римана*:

$$H_\infty^{(r)} = \zeta(r) = \sum_{k\geq 1} \frac{1}{k^r}. \tag{5}$$

Если r — четное целое число, то известно, что значение $\zeta(r)$ равно

$$H_\infty^{(r)} = \frac{1}{2}|B_r|\frac{(2\pi)^r}{r!}, \qquad \text{целое } r/2 \geq 1, \tag{6}$$

где B_r — это число Бернулли (см. раздел 1.2.11.2 и приложение А). В частности,

$$H_\infty^{(2)} = \frac{\pi^2}{6}, \quad H_\infty^{(4)} = \frac{\pi^4}{90}, \quad H_\infty^{(6)} = \frac{\pi^6}{945}, \quad H_\infty^{(8)} = \frac{\pi^8}{9450}. \tag{7}$$

Эти результаты получены Эйлером; подробное обсуждение данной темы, а также доказательства формул приводятся в *CMath*, §6.5.

А теперь рассмотрим несколько важных сумм, в которых участвуют гармонические числа. Во-первых,

$$\sum_{k=1}^{n} H_k = (n+1)H_n - n. \tag{8}$$

Это получается в результате простой замены индекса суммирования:

$$\sum_{k=1}^{n}\sum_{j=1}^{k}\frac{1}{j} = \sum_{j=1}^{n}\sum_{k=j}^{n}\frac{1}{j} = \sum_{j=1}^{n}\frac{n+1-j}{j}.$$

Формула (8) — частный случай суммы $\sum_{k=1}^{n}\binom{k}{m}H_k$, которую мы сейчас найдем. Для этого воспользуемся важным приемом, который называется суммированием по частям. Это очень полезный способ вычисления суммы $\sum a_k b_k$, когда величины $\sum a_k$ и $(b_{k+1}-b_k)$ имеют простой вид. В таком случае замечаем, что

$$\binom{k}{m}=\binom{k+1}{m+1}-\binom{k}{m+1},$$

и поэтому

$$\binom{k}{m}H_k=\binom{k+1}{m+1}\Big(H_{k+1}-\frac{1}{k+1}\Big)-\binom{k}{m+1}H_k;$$

следовательно,

$$\begin{aligned}\sum_{k=1}^{n}\binom{k}{m}H_k&=\Big(\binom{2}{m+1}H_2-\binom{1}{m+1}H_1\Big)+\cdots\\&\quad+\Big(\binom{n+1}{m+1}H_{n+1}-\binom{n}{m+1}H_n\Big)-\sum_{k=1}^{n}\binom{k+1}{m+1}\frac{1}{k+1}\\&=\binom{n+1}{m+1}H_{n+1}-\binom{1}{m+1}H_1-\frac{1}{m+1}\sum_{k=0}^{n}\binom{k}{m}+\frac{1}{m+1}\binom{0}{m}.\end{aligned}$$

Применяя соотношение 1.2.6–(11), получаем искомую формулу:

$$\sum_{k=1}^{n}\binom{k}{m}H_k=\binom{n+1}{m+1}\Big(H_{n+1}-\frac{1}{m+1}\Big).\tag{9}$$

(Вывод этой формулы и ее окончательный результат аналогичны вычислению интеграла

$$\int_1^n x^m\ln x\,dx=\frac{n^{m+1}}{m+1}\Big(\ln n-\frac{1}{m+1}\Big)+\frac{1}{(m+1)^2}$$

методом интегрирования по частям.)

И в завершение этого раздела рассмотрим сумму несколько иного вида,

$$\sum_k\binom{n}{k}x^kH_k,$$

которую для краткости временно обозначим через S_n. Находим, что

$$\begin{aligned}S_{n+1}=\sum_k\Big(\binom{n}{k}+\binom{n}{k-1}\Big)x^kH_k&=S_n+x\sum_{k\ge1}\binom{n}{k-1}x^{k-1}\Big(H_{k-1}+\frac{1}{k}\Big)\\&=S_n+xS_n+\frac{1}{n+1}\sum_{k\ge1}\binom{n+1}{k}x^k\end{aligned}$$

Отсюда $S_{n+1}=(x+1)S_n+\big((x+1)^{n+1}-1\big)/(n+1)$ и поэтому

$$\frac{S_{n+1}}{(x+1)^{n+1}}=\frac{S_n}{(x+1)^n}+\frac{1}{n+1}-\frac{1}{(n+1)(x+1)^{n+1}}.$$

Из данного равенства и того факта, что $S_1 = x$, следует

$$\frac{S_n}{(x+1)^n} = H_n - \sum_{k=1}^{n} \frac{1}{k(x+1)^k}.$$

Сумма справа является частной суммой бесконечного ряда для $\ln\bigl(1/(1-1/(x+1))\bigr) = \ln(1+1/x)$, этот ряд сходится при $x > 0$, разность между $ln(1+1/x)$ и частной суммой равна

$$\sum_{k>n} \frac{1}{k(x+1)^k} < \frac{1}{(n+1)(x+1)^{n+1}} \sum_{k\geq 0} \frac{1}{(x+1)^k} = \frac{1}{(n+1)(x+1)^n x}.$$

Таким образом, мы доказали следующую теорему.

Теорема А. *Если $x > 0$, то*

$$\sum_{k=1}^{n} \binom{n}{k} x^k H_k = (x+1)^n \left(H_n - \ln\left(1 + \frac{1}{x}\right)\right) + \epsilon,$$

где $0 < \epsilon < 1/\bigl(x(n+1)\bigr)$. ∎

УПРАЖНЕНИЯ

1. [*01*] Чему равны H_0, H_1 и H_2?

2. [*13*] Покажите, что, несколько видоизменив простое доказательство, которое было использовано в тексте для вывода неравенства $H_{2^m} \geq 1 + m/2$, можно показать, что $H_{2^m} \leq 1 + m$

3. [*M21*] Обобщите доказательство, использованное в предыдущем упражнении, и покажите, что для $r > 1$ сумма $H_n^{(r)}$ остается ограниченной для всех n Найдите верхнюю грань

▶ **4.** [*10*] Какие из следующих утверждений верны для всех положительных целых n. (a) $H_n < \ln n$; (b) $H_n > \ln n$, (c) $H_n > \ln n + \gamma$

5. [*15*] Пользуясь таблицами из приложения А, укажите значение H_{10000} с точностью до 15-го десятичного знака

6. [*M15*] Докажите, что гармонические числа непосредственно связаны с числами Стирлинга, которые рассматривались в предыдущем разделе, т е

$$H_n = \begin{bmatrix} n+1 \\ 2 \end{bmatrix} \Big/ n!$$

7. [*M21*] Пусть $T(m,n) = H_m + H_n - H_{mn}$ (a) Покажите, что если m или n возрастает, то $T(m,n)$ не возрастает (в предположении, что m и n положительны) (b) Вычислите минимальное и максимальное значения $T(m,n)$ для $m, n > 0$

8. [*HM18*] Сравните сумму (8) с $\sum_{k=1}^{n} \ln k$, найдите их разность как функцию от n

▶ **9.** [*M18*] Теорема А применима только для $x > 0$ Чему равна рассматриваемая сумма при $x = -1$?

10. [*M20*] (*Суммирование по частям*) В упр 1 2 4–42 и при выводе формулы (9) мы использовали частные случаи общего метода суммирования по частям Докажите общую формулу

$$\sum_{1\leq k<n} (a_{k+1} - a_k) b_k = a_n b_n - a_1 b_1 - \sum_{1\leq k<n} a_{k+1}(b_{k+1} - b_k)$$

▸ **11.** [*M21*] Пользуясь методом суммирования по частям, вычислите

$$\sum_{1<k\le n} \frac{1}{k(k-1)} H_k.$$

▸ **12.** [*M10*] Вычислите $H_\infty^{(1000)}$ с точностью по меньшей мере до 100-го десятичного знака.

13. [*M22*] Докажите тождество

$$\sum_{k=1}^{n} \frac{x^k}{k} = H_n + \sum_{k=1}^{n} \binom{n}{k} \frac{(x-1)^k}{k}.$$

(Обратите внимание на частный случай $x = 0$, который дает тождество, связанное с упр. 1.2.6-48.)

14. [*M22*] Покажите, что $\sum_{k=1}^{n} H_k/k = \frac{1}{2}(H_n^2 + H_n^{(2)})$, и вычислите $\sum_{k=1}^{n} H_k/(k+1)$.

▸ **15.** [*M23*] Выразите $\sum_{k=1}^{n} H_k^2$ через n и H_n.

16. [*18*] Выразите сумму $1 + \frac{1}{3} + \cdots + \frac{1}{2n-1}$ через гармонические числа.

17. [*M24*] (Э. Уоринг (E. Waring), 1782.) Пусть p — нечетное простое число. Покажите, что числитель H_{p-1} делится на p.

18. [*M33*] (Дж. Селфридж (J. Selfridge).) Какая наивысшая степень двойки делит числитель дроби $1 + \frac{1}{3} + \cdots + \frac{1}{2n-1}$?

▸ **19.** [*M30*] Перечислите все неотрицательные целые числа n, для которых H_n — целое число. [*Указание.* Если H_n имеет нечетный числитель и четный знаменатель, то оно не может быть целым числом.]

20. [*HM22*] Используя аналитический подход к решению задач суммирования (аналогичный тому, который привел нас к теореме A этого раздела), докажите следующее утверждение. Если $f(x) = \sum_{k\ge 0} a_k x^k$ и этот ряд сходится при $x = x_0$, то

$$\sum_{k\ge 0} a_k x_0^k H_k = \int_0^1 \frac{f(x_0) - f(x_0 y)}{1-y}\, dy.$$

21. [*M24*] Вычислите $\sum_{k=1}^{n} H_k/(n+1-k)$.

22. [*M28*] Вычислите $\sum_{k=0}^{n} H_k H_{n-k}$.

▸ **23.** [*HM20*] Рассмотрите функцию $\Gamma'(x)/\Gamma(x)$ и покажите с ее помощью, как можно естественным образом распространить H_n на нецелые значения n. Предвосхищая следующее упражнение, можете воспользоваться тем, что $\Gamma'(1) = -\gamma$.

24. [*HM21*] Покажите, что

$$xe^{\gamma x} \prod_{k\ge 1} \left(\left(1 + \frac{x}{k}\right) e^{-x/k}\right) = \frac{1}{\Gamma(x)}.$$

(Рассмотрите частные произведения этого бесконечного произведения.)

1.2.8. Числа Фибоначчи

Последовательность чисел

$$0,\ 1,\ 1,\ 2,\ 3,\ 5,\ 8,\ 13,\ 21,\ 34,\ \ldots, \tag{1}$$

каждый член которой является суммой двух предыдущих, играет важную роль приблизительно в десятке, казалось бы, несвязанных между собой алгоритмов, которые мы изучим несколько позже. Члены этой последовательности обозначаются

через F_n. Давайте формально определим их следующим образом:

$$F_0 = 0; \qquad F_1 = 1; \qquad F_{n+2} = F_{n+1} + F_n, \quad n \geq 0. \tag{2}$$

Эта знаменитая последовательность была представлена в 1202 году Леонардо Пизанским (Leonardo Pisano), которого иногда называют Леонардо Фибоначчи (Leonardo Fibonacci) (*Filius Bonaccii*, т. е. сын Боначчо). В его труде *Liber Abaci* ("Книга абака") содержится следующая задача: "Сколько пар кроликов можно получить от одной пары в год?". При этом используются такие предположения: каждая пара ежемесячно дает еще одну пару приплода, каждая новая пара становится способной к размножению в возрасте одного месяца и в течение этого года кролики не дохнут. Итак, через месяц у нас будет две пары кроликов, через два месяца — три пары, в следующем месяце первоначальная пара и пара, рожденная в первом месяце, дадут еще по паре кроликов, всего их станет пять и т. д.

Фибоначчи, без сомнения, был самым великим европейским математиком эпохи средневековья. Он изучил работу аль-Хорезми (от имени которого происходит слово "алгоритм"; см. раздел 1.1) и внес значительный вклад в развитие таких наук, как арифметика и геометрия. Труды Фибоначчи были переизданы в 1857 году [B. Boncompagni, *Scritti di Leonardo Pisano* (Rome, 1857–1862), 2 vols.; о числах F_n говорится в томе 1, с. 283–285]. Задача о кроликах, разумеется, была поставлена не для практического применения к биологии или теории о росте популяции; это было просто упражнение на сложение. Но, как ни странно, она до сих пор является прекрасным упражнением на сложение в курсе программирования (см. упр. 3). Фибоначчи писал: "Эту процедуру [сложение] можно выполнять для бесконечного числа месяцев".

Но еще до того, как Фибоначчи написал свой труд, последовательность $\langle F_n \rangle$ обсуждали индийские ученые в связи с проблемой стихосложения. Их издавна интересовали ритмические рисунки, которые образуются в результате чередования долгих и кратких слогов в стихах или сильных и слабых долей в музыке. Число таких ритмических рисунков, имеющих в целом n долей, равно F_{n+1}, поэтому Гопала (Gopāla) (до 1135 г.) и Хемачандра (Hemachandra) (ок. 1150 г.) в своих работах явно упоминали о числах 1, 2. 3, 5, 8, 13, 21, ... [См. P. Singh, *Historia Math.* **12** (1985), 229–244; см. также упр. 4.5.3–32.]

Эта же последовательность появляется и в работе Иоганна Кеплера (Johann Kepler) 1611 года, который размышлял о числах, встречающихся в природе [J. Kepler, *The Six-Cornered Snowflake* (Oxford: Clarendon Press, 1966), 21] (И. Кеплер "О шестиугольных снежинках" (М.: Наука, 1983)). Кеплер, по-видимому, не знал, что Фибоначчи уже упоминал эту последовательность в своих работах. Числа Фибоначчи часто встречаются в природе; вероятно, на это есть причины, аналогичные предположениям, которые мы сделали в задаче о кроликах. [См. работу Conway, Guy, *The Book of Numbers* (New York: Copernicus, 1996), 113–126, в которой этот вопрос освещается наиболее понятно и подробно.]

Первые признаки глубокой связи между числами F_n и алгоритмами были замечены в 1837 году, когда Э. Лежер (É. Léger) использовал последовательность Фибоначчи для изучения эффективности алгоритма Евклида. Он заметил, что если числа m и n в алгоритме 1.1E не превышают F_k, то шаг E2 будет выполнен максимум $k + 1$ раз. Это было первым практическим применением последовательности

Фибоначчи (см. теорему 4.5.3F.) В 70-х годах 19 века математик Э. Люка (É. Lucas) получил очень глубокие результаты, связанные с числами Фибоначчи; в частности, он использовал их для доказательства того, что состоящее из 39 цифр число $2^{127}-1$ является простым. Именно Люка дал последовательности $\langle F_n \rangle$ название "числа Фибоначчи", и с тех пор оно стало общепринятым.

Мы уже рассматривали последовательность Фибоначчи в разделе 1.2.1 (неравенство (3) и упр. 4) и выяснили, что $\phi^{n-2} \le F_n \le \phi^{n-1}$, если n — положительное целое, а

$$\phi = \tfrac{1}{2}(1+\sqrt{5}). \tag{3}$$

Вскоре мы увидим, что величина ϕ тесно связана с числами Фибоначчи.

Число ϕ и само имеет очень интересную историю. Евклид называл его отношением крайнего и среднего; отношение A к B равно отношению $A+B$ к A, если отношение A к B равно ϕ. В эпоху Возрождения это число называли божественной пропорцией; а в прошлом веке — золотым сечением. Многие художники и писатели говорили, что золотое сечение является наиболее эстетичным, и это мнение также справедливо с точки зрения эстетики компьютерного программирования. Об истории числа ϕ можно узнать из великолепной статьи H. S. M. Coxeter, "The Golden Section, Phyllotaxis, and Wythoff's Game", *Scripta Math.* **19** (1953), 135–143; см. также книгу Martin Gardner, *The 2nd Scientific American Book of Mathematical Puzzles and Diversions*, Chapter 8 (New York: Simon and Schuster, 1961) (Гарднер М. Математические головоломки и развлечения / Пер. с англ. — М.: Мир, 1971. — 25, 68 л.) Джордж Марковски (George Markowsky) опроверг некоторые распространенные мифы о числе ϕ в работе *College Math. J.* **23** (1992), 2–19. Тот факт, что отношение F_{n+1}/F_n приближается к ϕ при росте n, был известен средневековому ученому, специалисту в области счета Симону Жакобу (Simon Jacob), который умер в 1564 году [см. P. Schreiber, *Historia Math.* **22** (1995), 422–424].

Обозначения, используемые в этом разделе, не являются общепринятыми. Очень часто в специальной математической литературе вместо F_n пишут u_n, а вместо ϕ пишут τ. Наши обозначения почти повсеместно используются в популярной математической литературе (и в некоторых справочниках) и постепенно получают все более широкое распространение. Обозначение "ϕ" происходит от имени греческого скульптора Фидия (Phidias), который, говорят, часто применял золотое сечение в своей работе. Обозначение "F_n" используется потому, что именно так обозначена последовательность Фибоначчи в журнале *Fibonacci Quarterly*, в котором читатель может найти много интереснейших фактов, связанных с этой последовательностью. Хорошим примером классической работы, посвященной числам F_n, может служить глава 17 книги L. E. Dickson, *History of the Theory of Numbers* **1** (Carnegie Inst. of Washington, 1919).

Числа Фибоначчи удовлетворяют многим интересным тождествам; некоторые из них приведены в упражнениях к этому разделу. Приведем одно из наиболее часто "открываемых" соотношений, о котором Кеплер упоминал письме в 1608 году, хотя впервые оно было опубликовано Ж. Д. Кассини (J. D. Cassini) [*Histoire Acad. Roy. Paris* **1** (1680), 201]:

$$F_{n+1}F_{n-1} - F_n^2 = (-1)^n \tag{4}$$

Данное соотношение легко доказать по индукции. Но существует и более сложный метод. Он начинается с простого доказательства по индукции матричного тождества

$$\begin{pmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n.$$

Теперь, вычислив определители обеих частей этого равенства, получим (4).

Из формулы (4) следует, что числа F_n и F_{n+1} являются взаимно простыми, так как любой их общий делитель должен быть также делителем $(-1)^n$.

Из определения (2) непосредственно следует, что

$$F_{n+3} = F_{n+2} + F_{n+1} = 2F_{n+1} + F_n; \quad F_{n+4} = 3F_{n+1} + 2F_n.$$

В общем случае по индукции получаем, что

$$F_{n+m} = F_m F_{n+1} + F_{m-1} F_n \tag{6}$$

для любого положительного целого m.

Если в (6) взять m, кратное n, то по индукции находим, что

$$F_{nk} \text{ кратно } F_n.$$

Следовательно, каждое третье число последовательности Фибоначчи является четным, каждое четвертое кратно 3, каждое пятое кратно 5 и т. д.

На самом деле справедливо намного более сильное утверждение. Если наибольший общий делитель чисел m и n обозначить через $\gcd(m, n)$*, то можно сформулировать следующую удивительную теорему.

Теорема А (Э. Люка, 1876). *Некоторое целое число делит и F_m, и F_n тогда и только тогда, когда оно является делителем F_d, где $d = \gcd(m, n)$; в частности,*

$$\gcd(F_m, F_n) = F_{\gcd(m,n)}. \tag{7}$$

Доказательство. Для доказательства данной теоремы используется алгоритм Евклида. Из (6) следует, что любой общий делитель F_m и F_n является также делителем F_{n+m}; и наоборот, любой общий делитель F_{n+m} и F_n является делителем $F_m F_{n+1}$. Поскольку F_{n+1} и F_n взаимно просты, общий делитель F_{n+m} и F_n также делит F_m. Таким образом мы доказали, что для любого числа d

$$d \text{ делит } F_m \text{ и } F_n \text{ тогда и только тогда, когда } d \text{ делит } F_{m+n} \text{ и } F_n. \tag{8}$$

А теперь покажем, что *любая* последовательность $\langle F_n \rangle$, для которой $F_0 = 0$ и выполняется утверждение (8), удовлетворяет теореме А.

Сначала, воспользовавшись индукцией по k, обобщим утверждение (8) следующим образом:

$$d \text{ делит } F_m \text{ и } F_n \text{ тогда и только тогда, когда } d \text{ делит } F_{m+kn} \text{ и } F_n,$$

где k — любое неотрицательное целое число. Этот результат можно сформулировать более сжато:

$$d \text{ делит } F_{m \bmod n} \text{ и } F_n \text{ тогда и только тогда, когда } d \text{ делит } F_m \text{ и } F_n. \tag{9}$$

* Greatest common divisor — наибольший общий делитель. — *Прим. перев.*

Пусть r — остаток от деления числа m на n, т. е. $r = m \bmod n$. Тогда общие делители $\{F_m, F_n\}$ являются общими делителями $\{F_n, F_r\}$. Отсюда следует, что в процессе выполнения алгоритма 1.1E множество общих делителей чисел $\{F_m, F_n\}$ остается неизменным при изменении m и n. И наконец, при $r = 0$ общие делители — это просто делители чисел $F_0 = 0$ и $F_{\gcd(m,n)}$. ▮

Большинство важных результатов, связанных с числами Фибоначчи, можно вывести из формулы, в которой числа F_n выражаются через ϕ. Эту формулу мы сейчас и получим. Метод, которым мы воспользуемся, чрезвычайно важен, поэтому читателю, интересующемуся математикой, следует внимательно его изучить. Данный метод будет подробно рассматриваться в следующем разделе.

Для начала рассмотрим бесконечный ряд

$$\begin{aligned} G(z) &= F_0 + F_1 z + F_2 z^2 + F_3 z^3 + F_4 z^4 + \cdot \\ &= z + z^2 + 2z^3 + 3z^4 + \cdots. \end{aligned} \tag{10}$$

У нас нет никакой причины *заранее* ожидать, что этот бесконечный ряд сходится или что функция $G(z)$ вообще представляет какой-либо интерес. Но давайте будем оптимистами и посмотрим, что можно сказать о функции $G(z)$, если бесконечный ряд сходится. Преимущество подобного метода заключается в том, что $G(z)$ представляет *всю* последовательность Фибоначчи одновременно. Если же мы выясним, что представляет собой функция $G(z)$, то сможем определить ее коэффициенты. $G(z)$ называется *производящей функцией* для последовательности $\langle F_n \rangle$.

Теперь перейдем к исследованию функции $G(z)$:

$$\begin{aligned} zG(z) &= F_0 z + F_1 z^2 + F_2 z^3 + F_3 z^4 + \cdots, \\ z^2 G(z) &= \qquad F_0 z^2 + F_1 z^3 + F_2 z^4 + \cdots \end{aligned}$$

Вычитая два эти равенства из (10), получаем

$$\begin{aligned} (1 - z - z^2)G(z) = F_0 &+ (F_1 - F_0)z + (F_2 - F_1 - F_0)z^2 \\ &+ (F_3 - F_2 - F_1)z^3 + (F_4 - F_3 - F_2)z^4 + \cdots. \end{aligned}$$

Из определения F_n следует, что все члены, кроме второго, обращаются в нуль. Так как $(F_1 - F_0) = 1$, значение выражения в правой части равно z. Следовательно, если ряд (10) сходится, то

$$G(z) = z/(1 - z - z^2). \tag{11}$$

Эту функцию на самом деле *можно* представить в виде бесконечного ряда по степеням z (ряд Тейлора); отсюда следует, что коэффициенты степенного ряда для функции (11) должны быть числами Фибоначчи.

Теперь давайте выполним некоторые операции над $G(z)$, чтобы больше узнать о последовательности Фибоначчи. В формуле (11) знаменатель $1 - z - z^2$ представляет собой квадратный трехчлен; решив соответствующее квадратное уравнение, найдем два корня $\frac{1}{2}(-1 \pm \sqrt{5})$. После выполнения несложных преобразований можно разложить функцию $G(z)$ на элементарные дроби:

$$G(z) = \frac{1}{\sqrt{5}} \left(\frac{1}{1 - \phi z} - \frac{1}{1 - \hat{\phi} z} \right), \tag{12}$$

где

$$\hat{\phi} = 1 - \phi = \tfrac{1}{2}(1 - \sqrt{5}). \tag{13}$$

Величина $1/(1-\phi z)$ представляет собой сумму геометрической прогрессии $1 + \phi z + \phi^2 z^2 + \cdots$, поэтому

$$G(z) = \frac{1}{\sqrt{5}}\,(1 + \phi z + \phi^2 z^2 + \cdots - 1 - \hat{\phi} z - \hat{\phi}^2 z^2 - \cdots).$$

Коэффициенты при z^n должны быть равны F_n, поэтому

$$F_n = \frac{1}{\sqrt{5}}\,(\phi^n - \hat{\phi}^n). \tag{14}$$

Это важное выражение в замкнутой форме для чисел Фибоначчи было впервые получено в начале 18 века (См. D. Bernoulli, *Comment. Acad. Sci. Petrop.* **3** (1728), 85–100, §7; а также A. de Moivre, *Philos. Trans.* **32** (1922), 162–178. Де Муавр показал, как решать линейные рекуррентные соотношения общего вида. Он сделал это практически так же, как и мы при выводе формулы (14).)

Можно было бы просто привести формулу (14) и доказать ее по индукции. Но мы дали ее довольно длинное доказательство, в первую очередь, для того, чтобы показать, как *открывать* формулы с помощью метода производящих функций, который очень важен для решения многих задач.

Из формулы (14) можно вывести много важных фактов. Прежде всего, отметим, что $\hat{\phi}$ — это *отрицательное* число $(-0.61803\ldots)$, модуль которого меньше единицы. Поэтому с увеличением n последовательность $|\hat{\phi}^n|$ убывает. Фактически величина $\hat{\phi}^n/\sqrt{5}$ всегда настолько мала, что можно принять

$$F_n = \phi^n/\sqrt{5},\ \textit{округленное до ближайшего целого числа.} \tag{15}$$

Другие результаты можно непосредственно получить из определения $G(z)$, например

$$G(z)^2 = \frac{1}{5}\left(\frac{1}{(1-\phi z)^2} + \frac{1}{(1-\hat{\phi} z)^2} - \frac{2}{1-z-z^2}\right), \tag{16}$$

а коэффициент при z^n в формуле для $G(z)^2$ равен $\sum_{k=0}^{n} F_k F_{n-k}$. Отсюда получаем

$$\begin{aligned}\sum_{k=0}^{n} F_k F_{n-k} &= \tfrac{1}{5}\big((n+1)(\phi^n + \hat{\phi}^n) - 2F_{n+1}\big)\\ &= \tfrac{1}{5}\big((n+1)(F_n + 2F_{n-1}) - 2F_{n+1}\big)\\ &= \tfrac{1}{5}(n-1)F_n + \tfrac{2}{5}nF_{n-1}.\end{aligned} \tag{17}$$

(Второй шаг в этих выкладках следует из результата упр. 11.)

УПРАЖНЕНИЯ

1. [*10*] Решите первоначальную задачу, поставленную Леонардо Фибоначчи: сколько пар кроликов будет в наличии через год?

▸ **2.** [*20*] С помощью формулы (15) найдите приближенное значение F_{1000}. (Возьмите значения логарифмов из таблицы, приведенной в приложении А.)

3. [*25*] Напишите программу, которая вычисляет и выдает на печать числа Фибоначчи от F_1 до F_{1000} в десятичном виде. (В предыдущем упражнении был определен порядок чисел, с которыми придется иметь дело.)

▶ **4.** [*14*] Найдите все n, для которых $F_n = n$.

5. [*20*] Найдите все n, для которых $F_n = n^2$.

6. [*HM10*] Докажите тождество (5).

▶ **7.** [*15*] Если n не является простым числом, то и F_n не является простым числом (за одним исключением). Докажите это утверждение и найдите исключение.

8. [*15*] Во многих случаях удобно определить F_n для *отрицательных* n. Предположим, что $F_{n+2} = F_{n+1} + F_n$ для *всех* целых n. Проанализируйте эту возможность и ответьте на следующие вопросы. Чему равны F_{-1} и F_{-2}? Можно ли простым способом выразить F_{-n} через F_n?

9. [*M20*] Используя соглашения из упр. 8, определите, выполняются ли соотношения (4), (6), (14) и (15) при *любых* целых значениях нижних индексов.

10. [*15*] Выясните, будет ли значение $\phi^n/\sqrt{5}$ больше или меньше F_n.

11. [*M20*] Покажите, что $\phi^n = F_n\phi + F_{n-1}$ и $\hat{\phi}^n = F_n\hat{\phi} + F_{n-1}$ для *всех* целых n.

▶ **12.** [*M26*] Последовательность Фибоначчи "второго порядка" определяется соотношениями

$$\mathcal{F}_0 = 0, \quad \mathcal{F}_1 = 1, \quad \mathcal{F}_{n+2} = \mathcal{F}_{n+1} + \mathcal{F}_n + F_n.$$

Выразите $\mathcal{F}_n$ через F_n и F_{n+1}. [*Указание.* Воспользуйтесь производящими функциями.]

▶ **13.** [*M22*] Выразите следующие последовательности с помощью чисел Фибоначчи (r, s и c — заданные константы):

a) $a_0 = r$, $a_1 = s$; $a_{n+2} = a_{n+1} + a_n$, $n \geq 0$;
b) $b_0 = 0$, $b_1 = 1$; $b_{n+2} = b_{n+1} + b_n + c$, $n \geq 0$.

14. [*M28*] Пусть m — фиксированное положительное целое число. Найдите a_n, если

$$a_0 = 0, \quad a_1 = 1; \qquad a_{n+2} = a_{n+1} + a_n + \binom{n}{m} \quad \text{при } n \geq 0.$$

15. [*M22*] Пусть $f(n)$ и $g(n)$ — произвольные функции и пусть для $n \geq 0$

$$\begin{aligned} a_0 &= 0, \quad a_1 = 1, \quad a_{n+2} = a_{n+1} + a_n + f(n); \\ b_0 &= 0, \quad b_1 = 1, \quad b_{n+2} = b_{n+1} + b_n + g(n); \\ c_0 &= 0, \quad c_1 = 1, \quad c_{n+2} = c_{n+1} + c_n + xf(n) + yg(n) \end{aligned}$$

Выразите c_n через x, y, a_n, b_n и F_n.

▶ **16.** [*M20*] Числа Фибоначчи неявно присутствуют в треугольнике Паскаля. Покажите, что сумма биномиальных коэффициентов

$$\sum_{k=0}^{n} \binom{n-k}{k}$$

является числом Фибоначчи.

17. [*M24*] Используя соглашения из упр. 8, докажите следующее обобщение равенства (4): $F_{n+k}F_{m-k} - F_nF_m = (-1)^n F_{m-n-k}F_k$.

18. [*20*] Всегда ли $F_n^2 + F_{n+1}^2$ будет числом Фибоначчи?

▶ **19.** [*M27*] Чему равен $\cos 36°$?

20. [*M16*] Выразите сумму $\sum_{k=0}^{n} F_k$ с помощью чисел Фибоначчи.

21. [*M25*] Чему равна сумма $\sum_{k=0}^{n} F_k x^k$?

▸ **22.** [*M20*] Покажите, что $\sum_k \binom{n}{k} F_{m+k}$ является числом Фибоначчи.

23. [*M23*] Обобщая предыдущее упражнение, покажите, что $\sum_k \binom{n}{k} F_t^k F_{t-1}^{n-k} F_{m+k}$ всегда является числом Фибоначчи.

24. [*HM20*] Вычислите определитель порядка $n \times n$:

$$\begin{pmatrix} 1 & -1 & 0 & 0 & \ldots & 0 & 0 & 0 \\ 1 & 1 & -1 & 0 & \ldots & 0 & 0 & 0 \\ 0 & 1 & 1 & -1 & \ldots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \ldots & 1 & 1 & -1 \\ 0 & 0 & 0 & 0 & \ldots & 0 & 1 & 1 \end{pmatrix}$$

25. [*M21*] Покажите, что

$$2^n F_n = 2 \sum_{k \text{ odd}} \binom{n}{k} 5^{(k-1)/2}.$$

▸ **26.** [*M20*] Используя предыдущее упражнение, покажите, что $F_p \equiv 5^{(p-1)/2}$ (по модулю p), если p — нечетное простое число.

27. [*M20*] Используя предыдущее упражнение, покажите, что если p — простое число, не равное 5, то либо F_{p-1}, либо F_{p+1} (но не оба) кратно p.

28. [*M21*] Чему равно $F_{n+1} - \phi F_n$?

▸ **29.** [*M23*] (*Фибономиальные коэффициенты.*) Эдуард Люка определил величины

$$\binom{n}{k}_{\mathcal{F}} = \frac{F_n F_{n-1} \ldots F_{n-k+1}}{F_k F_{k-1} \ldots F_1} = \prod_{j=1}^{k} \left(\frac{F_{n-k+j}}{F_j} \right)$$

по аналогии с биномиальными коэффициентами. (a) Составьте таблицу значений $\binom{n}{k}_{\mathcal{F}}$ для $0 \le k \le n \le 6$. (b) Покажите, что $\binom{n}{k}_{\mathcal{F}}$ всегда является целым числом, поскольку

$$\binom{n}{k}_{\mathcal{F}} = F_{k-1} \binom{n-1}{k}_{\mathcal{F}} + F_{n-k+1} \binom{n-1}{k-1}_{\mathcal{F}}$$

▸ **30.** [*M38*] (Д. Джарден (D. Jarden) и Т. Моцкин (T. Motzkin).) Последовательность m-х степеней чисел Фибоначчи удовлетворяет рекуррентному соотношению, в котором каждый член зависит от предыдущих $m+1$ членов. Покажите, что

$$\sum_k \binom{m}{k}_{\mathcal{F}} (-1)^{\lceil (m-k)/2 \rceil} F_{n+k}^{m-1} = 0, \quad \text{если } m > 0$$

Например, при $m = 3$ получаем тождество $F_n^2 - 2F_{n+1}^2 - 2F_{n+2}^2 + F_{n+3}^2 = 0$.

31. [*M20*] Покажите, что $F_{2n}\phi \bmod 1 = 1 - \phi^{-2n}$ и $F_{2n+1}\phi \bmod 1 = \phi^{-2n-1}$.

32. [*M24*] Остаток от деления одного числа Фибоначчи на другое равен $\pm$ число Фибоначчи. Покажите, что по модулю F_n

$$F_{mn+r} \equiv \begin{cases} F_r, & \text{если } m \bmod 4 = 0; \\ (-1)^{r+1} F_{n-r}, & \text{если } m \bmod 4 = 1; \\ (-1)^n F_r, & \text{если } m \bmod 4 = 2; \\ (-1)^{r+1+n} F_{n-r}, & \text{если } m \bmod 4 = 3. \end{cases}$$

33. [*HM24*] Пусть $z = \pi/2 + i \ln \phi$. Покажите, что $\sin nz / \sin z = i^{1-n} F_n$.

▸ **34.** [*M24*] (*Система чисел Фибоначчи.*) Пусть запись $k \gg m$ означает, что $k \geq m+2$. Покажите, что для любого положительного целого n существует *единственное* представление $n = F_{k_1} + F_{k_2} + \cdots + F_{k_r}$, где $k_1 \gg k_2 \gg \cdots \gg k_r \gg 0$.

35. [*M24*] (*Система фи-чисел.*) Рассмотрим действительные числа, записанные с помощью цифр 0 и 1 по основанию ϕ (например, $(100.1)_\phi = \phi^2+\phi^{-1}$). Покажите, что существует бесконечное множество способов такого представления числа 1, например $1 = (.11)_\phi = (.011111\ldots)_\phi$. Но если потребовать, чтобы две единицы подряд не встречалось и чтобы это представление не заканчивалось бесконечной последовательностью 01010101..., то для каждого неотрицательного числа будет существовать единственное представление. Каким будет представление для целых чисел?

▸ **36.** [*M32*] (*Строки Фибоначчи.*) Пусть $S_1 =$ "a", $S_2 =$ "b" и $S_{n+2} = S_{n+1}S_n$, $n > 0$. Другими словами, S_{n+2} образуется путем помещения S_n справа от S_{n+1}. Имеем $S_3 =$ "ba", $S_4 =$ "bab", $S_5 =$ "$babba$" и т. д. Очевидно, что в строке S_n содержится F_n букв. Исследуйте свойства S_n. (Где встречаются две буквы подряд? Можете ли вы предсказать, какой будет k-я буква S_n? Какова плотность буквы b? И т. д.)

▸ **37.** [*M35*] (Р. Ю. Гаскел (R. E. Gaskell) и М. Дж. Виниган (M. J. Whinihan).) Двое играют в следующую игру. Имеется n фишек; первый игрок берет любое количество фишек (но только не все сразу). После этого момента игроки ходят по очереди, причем каждый из них берет одну или несколько фишек, но *не более чем двукратное количество фишек, взятых предыдущим игроком.* Выигрывает тот, кто заберет последнюю фишку. (Рассмотрим эту игру на конкретном примере. Пусть $n = 11$. Игрок A взял 3 фишки; значит, игрок B может забрать до 6 фишек, но он берет 1. Остается 7 фишек. Игрок A может взять 1 или 2 фишки; он берет 2. Теперь игрок B может взять до 4 фишек; он берет 1. Остается 4 фишки. На этот раз игрок A берет 1 фишку; теперь игрок B должен взять по меньшей мере одну фишку, поэтому на следующем ходе игрок A выигрывает.)

Сколько фишек следует взять первому игроку в начале игры, если первоначально имеется 1 000 фишек?

38. [*35*] Напишите такую компьютерную программу игры, описанной в предыдущем упражнении, чтобы она велась оптимальным образом.

39. [*M24*] Найдите выражение в замкнутой форме для a_n при условии, что $a_0 = 0$, $a_1 = 1$ и $a_{n+2} = a_{n+1} + 6a_n$ для $n \geq 0$.

40. [*M25*] Найдите решение рекуррентных соотношений

$$f(1) = 0; \quad f(n) = \min_{0<k<n} \max(1 + f(k), 2 + f(n-k)) \quad \text{для } n > 1$$

▸ **41.** [*M25*] (Юрий Матиясевич (Yuri Matiyasevich), 1990.) Пусть $f(x) = \lfloor x + \phi^{-1} \rfloor$. Докажите, что если $n = F_{k_1} + \cdots + F_{k_r}$ — это представление числа n в системе чисел Фибоначчи (см. упр. 34), то $F_{k_1+1} + \cdots + F_{k_r+1} = f(\phi n)$. Найдите аналогичную формулу для $F_{k_1-1} + \cdots + F_{k_r-1}$.

42. [*M26*] (Д. А. Кларнер (D. A. Klarner).) Покажите, что если m и n — неотрицательные целые числа, то существует единственная последовательность индексов $k_1 \gg k_2 \gg \cdots \gg k_r$, такая, что

$$m = F_{k_1} + F_{k_2} + \cdots + F_{k_r}, \qquad n = F_{k_1+1} + F_{k_2+1} + \cdots + F_{k_r+1}.$$

(Заметим, что k могут быть отрицательными, а r — нулем.)

1.2.9. Производящие функции

Каждый раз, когда необходимо получить информацию о последовательности чисел $\langle a_n \rangle = a_0, a_1, a_2, \ldots$, можно рассмотреть бесконечную сумму от "параметра" z

$$G(z) = a_0 + a_1 z + a_2 z^2 + \cdots = \sum_{n \geq 0} a_n z^n. \tag{1}$$

А теперь можно заняться исследованием свойств функции G. Данная функция характерна тем, что с ее помощью можно представить всю последовательность. Это очень важно, особенно если последовательность $\langle a_n \rangle$ была определена методом индукции (т. е. если a_n определяется через $a_0, a_1, \ldots, a_{n-1}$). Более того, с помощью методов дифференциального исчисления по функции $G(z)$ можно восстановить все члены последовательности a_0, a_1, ... (при условии, что бесконечный ряд (1) сходится для некоторого ненулевого z).

$G(z)$ называется *производящей функцией* для последовательности $a_0, a_1, a_2, \ldots$. Использование производящих функций открывает новые горизонты методов и приемов и существенно расширяет наши возможности при решении задач. Как уже упоминалось в предыдущем разделе, А. де Муавр ввел производящие функции, чтобы решить линейные рекуррентные соотношения в общем виде. Джеймс Стирлинг применил теорию де Муавра для решения более сложных рекуррентных соотношений и показал, как применять для этого не только арифметические операции, но также дифференцирование и интегрирование [*Methodus Differentialis* (London, 1730), Proposition 15]. Через несколько лет Л. Эйлер нашел новые способы использования производящих функций, например, при исследовании разбиений [*Commentarii Acad. Sci. Pet.* **13** (1741), 64–93; *Novi Comment. Acad. Sci. Pet.* **3** (1750), 125–169]. Дальнейшее развитие эти методы получили в классическом труде Пьера С. Лапласа (Pierre S. Laplace) *Théorie Analytique des Probabilités* (Paris, 1812).

Очень важен вопрос о сходимости бесконечного ряда (1). В любом учебнике по теории бесконечных рядов доказываются следующие утверждения.

a) Если ряд (1) сходится для некоторого $z = z_0$, то он сходится для всех z, таких, что $|z| < |z_0|$.

b) Этот ряд сходится для некоторого $z \neq 0$ тогда и только тогда, когда последовательность $\langle \sqrt[n]{|a_n|} \rangle$ ограничена. (Если это условие не выполняется, то можно получить сходящийся ряд для последовательности $\langle a_n/n! \rangle$ или другой последовательности, связанной с исходной.)

С другой стороны, во время работы с производящими функциями не всегда стоит беспокоиться о сходимости приведенного ряда, так как часто мы только исследуем возможные подходы к решению конкретной задачи. Найдя решение *некоторым* способом, каким бы нестрогим оно ни было, мы сможем обосновать его независимым образом. Например, в предыдущем разделе для вывода формулы (14) мы воспользовались производящей функцией. После того как данное равенство получено, уже не составляет труда доказать его по индукции и можно даже не упоминать, что оно было найдено с помощью производящей функции. Более того, можно показать, что большинство операций (если не все), выполняемых над производящими функциями, можно строго обосновать, не затрагивая вопроса о сходимости ряда. Например, см. работу E. T. Bell, *Trans. Amer. Math. Soc.* **25** (1923), 135–154; Ivan Niven, *AMM*

76 (1969), 871–889; Peter Henrici, *Applied and Computational Complex Analysis* **1** (Wiley, 1974), Chapter 1.

А теперь давайте рассмотрим основные методы работы с производящими функциями.

А. Сложение. Если $G(z)$ — производящая функция для $\langle a_n\rangle = a_0, a_1, \ldots$ и $H(z)$ — производящая функция для $\langle b_n\rangle = b_0, b_1, \ldots$, то $\alpha G(z) + \beta H(z)$ — производящая функция для $\langle \alpha a_n + \beta b_n\rangle = \alpha a_0 + \beta b_0, \alpha a_1 + \beta b_1, \ldots$:

$$\alpha \sum_{n\ge 0} a_n z^n + \beta \sum_{n\ge 0} b_n z^n = \sum_{n\ge 0} (\alpha a_n + \beta b_n) z^n. \tag{2}$$

В. Сдвиг. Если $G(z)$ — производящая функция для $\langle a_n\rangle = a_0, a_1, \ldots$, то $z^m G(z)$ — производящая функция для $\langle a_{n-m}\rangle = 0, \ldots, 0, a_0, a_1, \ldots$:

$$z^m \sum_{n\ge 0} a_n z^n = \sum_{n\ge m} a_{n-m} z^n. \tag{3}$$

В сумме справа суммирование можно распространить на все $n \ge 0$, если считать, что $a_n = 0$ *для любого отрицательного* n.

Аналогично $\big(G(z) - a_0 - a_1 z - \cdots - a_{m-1} z^{m-1}\big)/z^m$ — производящая функция для $\langle a_{n+m}\rangle = a_m, a_{m+1}, \ldots$:

$$z^{-m} \sum_{n\ge m} a_n z^n = \sum_{n\ge 0} a_{n+m} z^n. \tag{4}$$

В предыдущем разделе для решения задачи Фибоначчи мы комбинировали операции А и В: $G(z)$ была производящей функцией для $\langle F_n\rangle$, $zG(z)$ — для $\langle F_{n-1}\rangle$, $z^2G(z)$ — для $\langle F_{n-2}\rangle$, а $(1 - z - z^2)G(z)$ — для $\langle F_n - F_{n-1} - F_{n-2}\rangle$. Затем, так как разность $F_n - F_{n-1} - F_{n-2}$ равна нулю при $n \ge 2$, мы выяснили, что $(1 - z - z^2)G(z)$ — это многочлен. Аналогично производящая функция для любой *линейной рекуррентной* последовательности (т. е. такой последовательности, для которой $a_n = c_1 a_{n-1} + \cdots + c_m a_{n-m}$) является многочленом, деленным на $(1 - c_1 z - \cdots - c_m z^m)$.

Давайте рассмотрим простейший пример. Если $G(z)$ — производящая функция *для постоянной* последовательности $1, 1, 1, \ldots$, то $zG(z)$ порождает последовательность $0, 1, 1, \ldots$, поэтому $(1 - z)G(z) = 1$. В результате получаем простую, но очень важную формулу:

$$\frac{1}{1 - z} = 1 + z + z^2 + \cdots. \tag{5}$$

С. Умножение. Если $G(z)$ — производящая функция для $a_0, a_1, \ldots$ и $H(z)$ — производящая функция для $b_0, b_1, \ldots$, то

$$\begin{aligned} G(z)H(z) &= (a_0 + a_1 z + a_2 z^2 + \cdots)(b_0 + b_1 z + b_2 z^2 + \cdots) \\ &= (a_0 b_0) + (a_0 b_1 + a_1 b_0) z + (a_0 b_2 + a_1 b_1 + a_2 b_0) z^2 + \cdots. \end{aligned}$$

Следовательно, $G(z)H(z)$ — производящая функция для последовательности c_0, $c_1, \ldots$, где

$$c_n = \sum_{k=0}^{n} a_k b_{n-k}. \tag{6}$$

Соотношение (3) является частным случаем (6). Другой важный частный случай имеет место, когда все b_n равны единице:

$$\frac{1}{1-z}G(z) = a_0 + (a_0+a_1)z + (a_0+a_1+a_2)z^2 + \cdots. \tag{7}$$

Здесь мы получаем производящую функцию для частных сумм исходной последовательности.

Из соотношения (6) следует правило для произведения *трех* функций; $F(z)G(z)H(z)$ порождает последовательность $d_0, d_1, d_2, \ldots$, где

$$d_n = \sum_{\substack{i,j,k\geq 0\\ i+j+k=n}} a_i b_j c_k. \tag{8}$$

Общее правило для произведения *любого числа* функций (в тех случаях, когда это имеет смысл) выглядит следующим образом:

$$\prod_{j\geq 0}\sum_{k\geq 0} a_{jk}z^k = \sum_{n\geq 0} z^n \sum_{\substack{k_0,k_1,\ldots\geq 0\\ k_0+k_1+\cdots=n}} a_{0k_0}a_{1k_1}\ldots. \tag{9}$$

Когда в рекуррентном соотношении для некоторой последовательности содержатся биномиальные коэффициенты, часто возникает необходимость в получении производящей функции для последовательности $c_0, c_1, \ldots$, которая определяется формулой

$$c_n = \sum_k \binom{n}{k} a_k b_{n-k}. \tag{10}$$

В этом случае, как правило, лучше воспользоваться производящими функциями последовательностей $\langle a_n/n!\rangle$, $\langle b_n/n!\rangle$, $\langle c_n/n!\rangle$, поскольку

$$\left(\frac{a_0}{0!}+\frac{a_1}{1!}z+\frac{a_2}{2!}z^2+\cdots\right)\left(\frac{b_0}{0!}+\frac{b_1}{1!}z+\frac{b_2}{2!}z^2+\cdots\right) = \left(\frac{c_0}{0!}+\frac{c_1}{1!}z+\frac{c_2}{2!}z^2+\cdots\right), \tag{11}$$

где c_n определяется формулой (10).

D. Замена переменной z. Очевидно, что $G(cz)$ — производящая функция для последовательности $a_0, ca_1, c^2a_2, \ldots$. В частности, производящей функцией для последовательности $1, c, c^2, c^3, \ldots$ является $1/(1-cz)$.

Воспользуемся известным приемом для извлечения членов ряда через один:

$$\begin{aligned} \tfrac{1}{2}\big(G(z)+G(-z)\big) &= a_0 + a_2z^2 + a_4z^4 + \cdots,\\ \tfrac{1}{2}\big(G(z)-G(-z)\big) &= a_1z + a_3z^3 + a_5z^5 + \cdots \end{aligned} \tag{12}$$

Извлекая комплексные корни из единицы, можно развивать эту идею и выбирать каждый m-й член. Пусть $\omega = e^{2\pi i/m} = \cos(2\pi/m) + i\sin(2\pi/m)$. Тогда

$$\sum_{n \bmod m = r} a_n z^n = \frac{1}{m}\sum_{0\leq k<m} \omega^{-kr}G(\omega^k z), \quad 0\leq r<m. \tag{13}$$

(См. упр. 14.) Например, если $m=3$ и $r=1$, то $\omega = -\frac{1}{2}+\frac{\sqrt{3}}{2}i$ (комплексный кубический корень из единицы). Отсюда следует, что

$$a_1z + a_4z^4 + a_7z^7 + \cdots = \tfrac{1}{3}\big(G(z)+\omega^{-1}G(\omega z)+\omega^{-2}G(\omega^2 z)\big).$$

E. Дифференцирование и интегрирование. С помощью методов дифференциального и интегрального исчислений можно выполнять над производящими функциями дополнительные операции. Если $G(z)$ определяется формулой (1), то ее производная равна

$$G'(z) = a_1 + 2a_2 z + 3a_3 z^2 + \cdots = \sum_{k\ge 0}(k+1)a_{k+1}z^k. \tag{14}$$

Производящей функцией для последовательности $\langle na_n\rangle$ является $zG'(z)$. Следовательно, выполняя операции над производящей функцией, мы можем найти производящую функцию для последовательности, полученной из предыдущей умножением каждого члена a_n на многочлен от n.

Обратный процесс, т. е. интегрирование, дает еще одну полезную операцию:

$$\int_0^z G(t)\,dt = a_0 z + \frac{1}{2}a_1 z^2 + \frac{1}{3}a_2 z^3 + \cdots = \sum_{k\ge 1}\frac{1}{k}a_{k-1}z^k. \tag{15}$$

В качестве частных случаев возьмем производную и интеграл от функции (5):

$$\frac{1}{(1-z)^2} = 1 + 2z + 3z^2 + \cdots = \sum_{k\ge 0}(k+1)z^k; \tag{16}$$

$$\ln\frac{1}{1-z} = z + \frac{1}{2}z^2 + \frac{1}{3}z^3 + \cdots = \sum_{k\ge 1}\frac{1}{k}z^k. \tag{17}$$

Сопоставляя (17) и (7), найдем производящую функцию для последовательности гармонических чисел:

$$\frac{1}{1-z}\ln\frac{1}{1-z} = z + \frac{3}{2}z^2 + \frac{11}{6}z^3 + \cdots = \sum_{k\ge 0}H_k z^k.$$

F. Известные производящие функции. В каждом случае, когда функция разлагается в степенной ряд, мы, по сути, получаем производящую функцию для некоторой последовательности. Эти специальные функции могут быть очень полезны в сочетании с операциями, описанными выше. Ниже приведены самые важные случаи разложения в степенной ряд.

i) *Биномиальная теорема.*

$$(1+z)^r = 1 + rz + \frac{r(r-1)}{2}z^2 + \cdots = \sum_{k\ge 0}\binom{r}{k}z^k. \tag{19}$$

Если r — неотрицательное целое число, то получаем частный случай, который уже отражен в соотношениях (5) и (16):

$$\frac{1}{(1-z)^{n+1}} = \sum_{k\ge 0}\binom{-n-1}{k}(-z)^k = \sum_{k\ge 0}\binom{n+k}{n}z^k. \tag{20}$$

Существует также обобщенная формула, доказанная в упр. 1.2.6–25:

$$x^r = 1 + rz + \frac{r(r-2t-1)}{2}z^2 + \cdots = \sum_{k\ge 0}\binom{r-kt}{k}\frac{r}{r-kt}z^k; \tag{21}$$

здесь x — непрерывная функция от z, которая является решением уравнения $x^{t+1} = x^t + z$, где $x = 1$ при $z = 0$.

ii) *Экспоненциальный ряд.*

$$\exp z = e^z = 1 + z + \frac{1}{2!}z^2 + \cdots = \sum_{k\geq 0} \frac{1}{k!}z^k. \tag{22}$$

В общем случае имеем следующую формулу, содержащую числа Стирлинга:

$$(e^z - 1)^n = z^n + \frac{1}{n+1}\left\{ {n+1 \atop n} \right\} z^{n+1} + \cdots = n! \sum_k \left\{ {k \atop n} \right\} z^k/k!. \tag{23}$$

iii) *Логарифмический ряд* (см. (17) и (18)).

$$\ln(1+z) = z - \frac{1}{2}z^2 + \frac{1}{3}z^3 - \cdots = \sum_{k\geq 1} \frac{(-1)^{k+1}}{k} z^k, \tag{24}$$

$$\frac{1}{(1-z)^{m+1}} \ln\Big(\frac{1}{1-z}\Big) = \sum_{k\geq 1} (H_{m+k} - H_m) \binom{m+k}{k} z^k. \tag{25}$$

С помощью чисел Стирлинга можно получить более общее соотношение (как в (23)):

$$\Big(\ln \frac{1}{1-z}\Big)^n = z^n + \frac{1}{n+1}\left[{n+1 \atop n} \right] z^{n+1} + \cdots = n! \sum_k \left[{k \atop n} \right] z^k/k!. \tag{26}$$

Еще более обобщенные формулы, включающие суммы гармонических чисел, можно найти в статьях D. A. Zave, *Inf. Proc. Letters* **5** (1976), 75–77; J. Spieß, *Math. Comp.* **55** (1990), 839–863.

iv) *Другие ряды.*

$$z(z+1)\ldots(z+n-1) = \sum_k \left[{n \atop k} \right] z^k, \tag{27}$$

$$\frac{z^n}{(1-z)(1-2z)\ldots(1-nz)} = \sum_k \left\{ {k \atop n} \right\} z^k, \tag{28}$$

$$\frac{z}{e^z - 1} = 1 - \frac{1}{2}z + \frac{1}{12}z^2 + \cdots = \sum_{k\geq 0} \frac{B_k z^k}{k!}. \tag{29}$$

Коэффициенты B_k в последней формуле — это *числа Бернулли*; более подробно они будут обсуждаться в разделе 1.2.11.2. Таблица чисел Бернулли приведена в приложении A.

Следующее тождество, аналогичное (21), будет доказано в упр. 2.3.4.4–29:

$$x^r = 1 + rz + \frac{r(r+2t)}{2}z^2 + \cdots = \sum_{k\geq 0} \frac{r(r+kt)^{k-1}}{k!} z^k; \tag{30}$$

здесь x — непрерывная функция от z, которая является решением уравнения $x = e^{zx^t}$, где $x = 1$ при $z = 0$. Важные обобщения формул (21) и (30) обсуждаются в упр. 4.7–22.

G. Представление коэффициента. Для коэффициента при z^n в выражении для $G(z)$ часто удобно использовать запись

$$[z^n]G(z). \tag{31}$$

Например, если $G(z)$ — производящая функция, определяемая формулой (1), то $[z^n]G(z) = a_n$ и $[z^n]G(z)/(1-z) = \sum_{k=0}^{n} a_k$. Одним из самых фундаментальных результатов теории комплексного переменного является формула О. Л. Коши (A. L. Cauchy) [*Exercices de Math.* **1** (1826), 95–113 = *Œuvres* (2) **6**, 124–145, Eq. (11)], по которой любой нужный коэффициент можно представить в виде интеграла по контуру

$$[z^n]\,G(z) = \frac{1}{2\pi i}\oint_{|z|=r} \frac{G(z)\,dz}{z^{n+1}}, \tag{32}$$

если $G(z)$ сходится для $z = z_0$ и $0 < r < |z_0|$. Главная идея состоит в том, что интеграл $\oint_{|z|=r} z^m\,dz$ равен нулю для всех целых m, за исключением $m = -1$, когда интеграл равен

$$\int_{-\pi}^{\pi} (re^{i\theta})^{-1}\,d(re^{i\theta}) = i\int_{-\pi}^{\pi} d\theta = 2\pi i.$$

Формула (32) особенно важна в случае, когда мы хотим изучить поведение коэффициента.

И в заключение раздела вернемся к задаче, которая была лишь частично решена в разделе 1.2.3. Из формулы 1.2.3–(13) и упр. 1.2.3–29 следует, что

$$\sum_{1\le i\le j\le n} x_i x_j = \frac{1}{2}\left(\sum_{k=1}^{n} x_k\right)^2 + \frac{1}{2}\left(\sum_{k=1}^{n} x_k^2\right);$$

$$\sum_{1\le i\le j\le k\le n} x_i x_j x_k = \frac{1}{6}\left(\sum_{k=1}^{n} x_k\right)^3 + \frac{1}{2}\left(\sum_{k=1}^{n} x_k\right)\left(\sum_{k=1}^{n} x_k^2\right) + \frac{1}{3}\left(\sum_{k=1}^{n} x_k^3\right)$$

Рассмотрим общий случай. Пусть есть n чисел $x_1, x_2, \ldots, x_n$ и нужно найти сумму

$$h_m = \sum_{1\le j_1\le\cdots\le j_m\le n} x_{j_1}\ldots x_{j_m}. \tag{33}$$

Требуется, если возможно, выразить эту сумму через S_1, S_2, …, S_m, где

$$S_j = \sum_{k=1}^{n} x_k^j \tag{34}$$

представляет собой сумму j-х степеней. Используя эту более компактную запись, можно переписать приведенные выше формулы следующим образом:

$$h_2 = \tfrac{1}{2}S_1^2 + \tfrac{1}{2}S_2; \qquad h_3 = \tfrac{1}{6}S_1^3 + \tfrac{1}{2}S_1S_2 + \tfrac{1}{3}S_3.$$

Чтобы решить задачу, рассмотрим производящую функцию

$$H(z) = 1 + h_1 z + h_2 z^2 + \cdots = \sum_{k\ge 0} h_k z^k \tag{35}$$

Согласно правилу умножения рядов находим

$$G(z) = (1 + x_1 z + x_1^2 z^2 + \cdots)(1 + x_2 z + x_2^2 z^2 + \cdots)\ldots(1 + x_n z + x_n^2 z^2 + \cdots)$$
$$= \frac{1}{(1 - x_1 z)(1 - x_2 z)\ldots(1 - x_n z)}. \tag{36}$$

Таким образом, $G(z)$ — это величина, обратная многочлену. Во многих случаях бывает полезно прологарифмировать произведение; сделав это и воспользовавшись формулой (17), получим

$$\ln G(z) = \ln\frac{1}{1 - x_1 z} + \cdots + \ln\frac{1}{1 - x_n z}$$
$$= \left(\sum_{k\ge 1}\frac{x_1^k z^k}{k}\right) + \cdots + \left(\sum_{k\ge 1}\frac{x_n^k z^k}{k}\right) = \sum_{k\ge 1}\frac{S_k z^k}{k}. \tag{37}$$

Таким образом, мы выразили $\ln G(z)$ через S_k, $k \ge 1$. Теперь для получения окончательного ответа осталось найти разложение $G(z)$ в степенной ряд с помощью (22) и (9):

$$G(z) = e^{\ln G(z)} = \exp\Bigl(\sum_{k\ge 1}\frac{S_k z^k}{k}\Bigr) = \prod_{k\ge 1} e^{S_k z^k/k}$$
$$= \left(1 + S_1 z + \frac{S_1^2 z^2}{2!} + \cdots\right)\left(1 + \frac{S_2 z^2}{2} + \frac{S_2^2 z^4}{2^2\cdot 2!} + \cdots\right)\ldots$$
$$= \sum_{m\ge 0}\Biggl(\sum_{\substack{k_1,k_2,\ \ldots,k_m\ge 0\\ k_1+2k_2+\ \ldots\ +mk_m=m}} \frac{S_1^{k_1}}{1^{k_1}k_1!}\,\frac{S_2^{k_2}}{2^{k_2}k_2!}\cdots\frac{S_m^{k_m}}{m^{k_m}k_m!}\Biggr) z^m. \tag{38}$$

Величина в круглых скобках — h_m. Эта внушительная сумма при внимательном рассмотрении оказывается не такой уж сложной. Число членов для конкретного значения m равно $p(m)$, т. е. числу разбиений m (см. раздел 1.2.1). Например, одним из разбиений числа 12 является

$$12 = 5 + 2 + 2 + 2 + 1;$$

это соответствует некоторому решению уравнения $k_1 + 2k_2 + \cdots + 12k_{12} = 12$, где k_j — количество слагаемых в разбиении, равных j. В нашем примере $k_1 = 1$, $k_2 = 3$, $k_5 = 1$, а все остальные k_j равны нулю, поэтому получаем член

$$\frac{S_1}{1^1 1!}\,\frac{S_2^3}{2^3 3!}\,\frac{S_5}{5^1 1!} = \frac{1}{240} S_1 S_2^3 S_5,$$

который является частью выражения для h_{12}. Дифференцируя (37), нетрудно получить рекуррентное соотношение

$$h_n = \frac{1}{n}(S_1 h_{n-1} + S_2 h_{n-2} + \cdots + S_n h_0), \qquad n \ge 1. \tag{39}$$

Замечательное введение в теорию применений производящих функций можно найти в книге G. Pólya, *On picture writing*, *AMM* **63** (1956), 689–697; этот подход

был использован в *CMath*, Chapter 7. [См. также книгу H. S. Wilf, *Generatingfunctionology*, second edition (Academic Press, 1994).]

Производящая функция — это бельевая веревка,
на которую мы вывешиваем последовательность чисел
для всеобщего обозрения

— Г. С. ВИЛЬФ (H. S. WILF) (1989)

УПРАЖНЕНИЯ

1. [*M12*] Найдите производящую функцию для последовательности 2, 5, 13, 35, ... $\langle 2^n + 3^n \rangle$.

▶ **2.** [*M13*] Докажите формулу (11).

3. [*HM21*] Продифференцируйте производящую функцию (18) для последовательности $\langle H_n \rangle$ и сравните результат с производящей функцией для последовательности $\langle \sum_{k=0}^{n} H_k \rangle$. Какую связь между ними вы обнаружили?

4. [*M01*] Объясните, почему (19) является частным случаем (21).

5. [*M20*] Докажите (23) индукцией по n

▶ **6.** [*HM15*] Найдите производящую функцию для последовательности

$$\left\langle \sum_{0<k<n} \frac{1}{k(n-k)} \right\rangle,$$

продифференцируйте ее и выразите коэффициенты через гармонические числа.

7. [*M15*] Проверьте все этапы вывода соотношения (38)

8. [*M23*] Найдите производящую функцию для последовательности $p(n)$ — числа разбиений целого n.

9. [*M11*] Используя обозначения из соотношений (34) и (35), выразите h_4 через S_1, S_2, S_3 и S_4

▶ **10.** [*M25*] *Элементарная симметричная функция* определяется по формуле

$$a_m = \sum_{1 \le j_1 < \quad < j_m \le n} x_{j_1} \cdots x_{j_m}$$

(Это то же самое, что и h_m из (33), только при суммировании не допускается равенство индексов) Найдите производящую функцию для последовательности a_m и выразите a_m через S_j, определяемые формулой (34) Выпишите формулы для a_1, a_2, a_3 и a_4

▶ **11.** [*M25*] Соотношение (39) можно также использовать для того, чтобы выразить S_k через h_k находим $S_1 = h_1$, $S_2 = 2h_2 - h_1^2$, $S_3 = 3h_3 - 3h_1h_2 + h_1^3$ и т д Чему равен коэффициент при $h_1^{k_1} h_2^{k_2} \quad h_m^{k_m}$ в таком представлении S_m, если $k_1 + 2k_2 + \quad + mk_m = m$?

▶ **12.** [*M20*] Пусть у нас есть последовательность с двумя индексами a_{mn}, где $m, n = 0, 1, \ldots$ Покажите, что эту последовательность можно представить с помощью *одной* производящей функции двух переменных, и найдите производящую функцию для последовательности $a_{mn} = \binom{n}{m}$

13. [*HM22*] *Преобразованием Лапласа* функции $f(x)$ называется функция

$$\mathrm{L}f(s) = \int_0^\infty e^{-st} f(t)\, dt$$

Пусть $a_0, a_1, a_2, \ldots$ — бесконечная последовательность, производящая функция которой сходится, и пусть $f(x)$ — ступенчатая функция $\sum_k a_k [0 \le k \le x]$. Выразите преобразование Лапласа функции $f(x)$ через производящую функцию G заданной последовательности.

14. [*HM21*] Докажите соотношение (13).

15. [*M28*] Рассмотрим функцию $H(w) = \sum_{n\ge 0} G_n(z) w^n$. Найдите замкнутую форму для производящей функции

$$G_n(z) = \sum_{k=0}^{n} \binom{n-k}{k} z^k = \sum_{k=0}^{n} \binom{2k-n-1}{k} (-z)^k.$$

16. [*M22*] Найдите простую формулу для производящей функции $G_{nr}(z) = \sum_k a_{nkr} z^k$, где a_{nkr} — количество способов выбора k объектов из n при условии, что каждый объект можно выбрать максимум r раз. (При $r = 1$ получаем $\binom{n}{k}$ способов, а при $r \ge k$ — число сочетаний с повторениями, как в упр 1.2.6–60.)

17. [*M25*] Найдите коэффициенты в разложении функции $1/(1-z)^w$ в *двойной* степенной ряд по z и w

▸ **18.** [*M25*] Для заданных положительных целых чисел n и r найдите простые формулы для следующих сумм: (a) $\sum_{1\le k_1<k_2< \ <k_r\le n} k_1 k_2 \ . k_r$, (b) $\sum_{1\le k_1\le k_2\le \ \le k_r\le n} k_1 k_2 \ldots k_r$. (Например, для $n = 3$ и $r = 2$ эти суммы соответственно равны $1 \ \ 2 + 1 \cdot 3 + 2 \cdot 3$ и $1 \cdot 1 + 1 \cdot 2 + 1 \cdot 3 + 2 \cdot 2 + 2 \cdot 3 + 3 \cdot 3$.)

19. [*HM32*] (К Ф Гаусс (C F Gauss), 1812) Хорошо известны суммы следующих бесконечных рядов:

$$1 - \frac{1}{2} + \frac{1}{3} - \frac{1}{4} + \cdots = \ln 2; \quad 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \cdot \ \cdot = \frac{\pi}{4},$$

$$1 - \frac{1}{4} + \frac{1}{7} - \frac{1}{10} + \cdots = \frac{\pi\sqrt{3}}{9} + \frac{1}{3}\ln 2$$

С помощью определения

$$H_x = \sum_{n\ge 1} \Big(\frac{1}{n} - \frac{1}{n+x} \Big),$$

данного в ответе к упр. 1.2.7–24, эти ряды можно переписать следующим образом соответственно:

$$1 - \frac{1}{2} H_{1/2}; \qquad \frac{2}{3} - \frac{1}{4} H_{1/4} + \frac{1}{4} H_{3/4}, \qquad \frac{3}{4} - \frac{1}{6} H_{1/6} + \frac{1}{6} H_{2/3}.$$

Докажите, что в общем случае значение $H_{p/q}$ равно

$$\frac{q}{p} - \frac{\pi}{2} \cot \frac{p}{q}\pi - \ln 2q + 2 \sum_{0<k<q/2} \cos \frac{2pk}{q}\pi \cdot \ln \sin \frac{k}{q}\pi,$$

где p и q — целые числа и $0 < p < q$ [*Указание.* По теореме Абеля эта сумма равна

$$\lim_{x\to 1-} \sum_{n\ge 1} \Big(\frac{1}{n} - \frac{1}{n+p/q} \Big) x^{p+nq}.$$

С помощью соотношения (13) выразите этот степенной ряд таким образом, чтобы можно было вычислить предел]

20. [*M21*] Для каких коэффициентов c_{mk} справедливо равенство

$$\sum_{n\ge 0} n^m z^n = \sum_{k=0}^{m} c_{mk} z^k / (1-z)^{k+1}?$$

21. [*HM30*] Найдите производящую функцию для последовательности $\langle n! \rangle$ и исследуйте свойства этой функции.

22. [*M21*] Найдите производящую функцию $G(z)$, для которой

$$[z^n]\,G(z) = \sum_{k_0+2k_1+4k_2+8k_3+\cdots=n} \binom{r}{k_0}\binom{r}{k_1}\binom{r}{k_2}\binom{r}{k_3}\cdots.$$

23. [*M33*] (Л. Карлитц (L. Carlitz).) (a) Докажите, что для всех целых чисел $m \geq 1$ существуют многочлены $f_m(z_1,\ldots,z_m)$ и $g_m(z_1,\ldots,z_m)$, такие, что формула

$$\sum_{k_1,\ \ldots,k_m\geq 0} \binom{r}{n-k_1}\binom{k_1}{n-k_2}\cdots\binom{k_{m-1}}{n-k_m} z_1^{k_1}\ldots z_m^{k_m} = f_m(z_1,\ldots,z_m)^{n-r}\,g_m(z_1,\ldots,z_m)^r$$

превращается в тождество для всех целых чисел $n \geq r \geq 0$.

(b) Обобщая упр. 15, найдите замкнутую форму для суммы

$$S_n(z_1,\ldots,z_m) = \sum_{k_1,\ \ldots,k_m\geq 0} \binom{k_1}{n-k_2}\binom{k_2}{n-k_3}\cdots\binom{k_m}{n-k_1} z_1^{k_1}\ldots z_m^{k_m}$$

выразив ее через функции f_m и g_m из п. (a).

(c) Найдите простое выражение для $S_n(z_1,\ldots,z_m)$, если $z_1 = \cdots = z_m = z$.

24. [*M22*] Докажите, что для любой производящей функции $G(z)$

$$\sum_k \binom{m}{k} [z^{n-k}]\,G(z)^k = [z^n]\,(1+zG(z))^m.$$

Вычислите обе части этого тождества, если $G(z)$ равна. (a) $1/(1-z)$; (b) $(e^z-1)/z$.

▸ **25.** [*M23*] Вычислите сумму $\sum_k \binom{n}{k}\binom{2n-2k}{n-k}(-2)^k$, упростив эквивалентную формулу $\sum_k [w^k]\,(1-2w)^n\,[z^{n-k}]\,(1+z)^{2n-2k}$.

26. [*M40*] Найдите обобщение обозначения (31), согласно которому можно, например, записать $[z^2-2z^5]\,G(z) = a_2 - 2a_5$, где $G(z)$ задается формулой (1).

1.2.10. Анализ алгоритма

Пришло время применить некоторые методы, рассмотренные в предыдущих разделах, к изучению типичного алгоритма.

Алгоритм М (*Нахождение максимума*). Для заданных n элементов $X[1], X[2], \ldots, X[n]$ необходимо найти такие величины m и j, что $m = X[j] = \max_{1\leq i\leq n} X[i]$, где j — наибольший индекс, удовлетворяющий этому соотношению.

М1. [Инициализация.] Положим $j \leftarrow n$, $k \leftarrow n-1$, $m \leftarrow X[n]$. (Во время выполнения алгоритма будем иметь $m = X[j] = \max_{k<i\leq n} X[i]$.)

М2. [Все проверено?] Если $k = 0$, то работа алгоритма заканчивается.

М3. [Сравнение.] Если $X[k] \leq m$, перейти к шагу М5.

М4. [Замена m.] Положим $j \leftarrow k$, $m \leftarrow X[k]$. (Это значение m является новым текущим максимумом.)

М5. [Уменьшение k.] Уменьшим k на единицу и вернемся к шагу М2. ▮

Может показаться, что данный алгоритм настолько тривиален, что его вообще не стоит подробно анализировать. Но на самом деле это хороший пример, показывающий, как можно исследовать более сложные алгоритмы. Анализ алгоритмов имеет

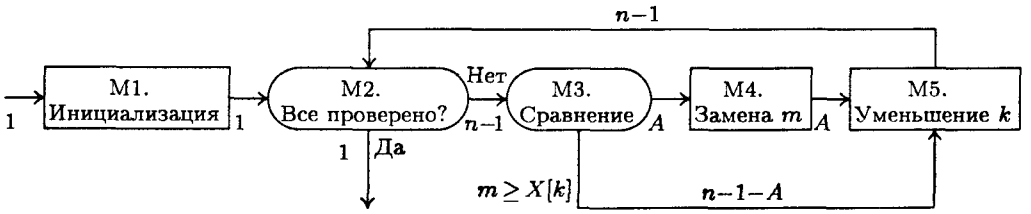


Рис. 9. Алгоритм М Подписи к стрелкам показывают, сколько раз осуществляется проход каждого пути Заметьте, что должен выполняться первый закон Кирхгофа, который в данном случае выглядит так: число входов в каждый узел должно равняться числу выходов из него.

очень большое значение в программировании, потому что, как правило, существует несколько алгоритмов решения конкретной задачи и необходимо знать, какой из них наилучший.

Для алгоритма М требуется фиксированный объем памяти, поэтому будем анализировать только время, необходимое для его выполнения. Для этого подсчитаем, сколько раз выполняется каждый шаг (рис. 9).

Номер шага	Количество выполнений
M1	1
M2	n
M3	$n-1$
M4	A
M5	$n-1$

Зная, сколько раз выполняется каждый шаг, можно определить время работы алгоритма на конкретном компьютере.

В приведенной выше таблице неизвестна только величина A, которая определяет, сколько раз необходимо изменить значение текущего максимума. Чтобы довести анализ до конца, исследуем эту любопытную величину A.

Этот анализ обычно заключается в нахождении *минимального* (для оптимистов), *максимального* (для пессимистов), *среднего* (для специалистов по теории вероятностей) значения A и его *среднего квадратичного отклонения* (данная характеристика показывает, насколько близким к среднему может оказаться значение).

Минимальное значение A равно нулю; это будет в случае, если

$$X[n] = \max_{1\le k\le n} X[k].$$

Максимальное значение A равно $n-1$; это будет в случае, если

$$X[1] > X[2] > \cdots > X[n].$$

Таким образом, среднее значение лежит между 0 и $n-1$. Будет ли это $\frac{1}{2}n$? Или $\sqrt{n}$? Чтобы ответить на поставленный вопрос, нужно выяснить, что понимается под средним. А чтобы правильно определить среднее, сделаем некоторые предположения, касающиеся характеристик входных данных $X[1], X[2], \ldots, X[n]$. *Будем считать, что $X[k]$ — различные значения и что все $n!$ перестановок этих значений равновероятны.* (В большинстве случаев это разумное допущение, но,

как вы увидите из упражнений к данному разделу, анализ можно проводить также, исходя из других предположений.)

Производительность алгоритма M не зависит от самих величин $X[k]$; имеет значение только относительный порядок их расположения. Например, если $n = 3$, предполагается, что следующие шесть возможностей равновероятны.

Ситуация	Значение A	Ситуация	Значение A
$X[1] < X[2] < X[3]$	0	$X[2] < X[3] < X[1]$	1
$X[1] < X[3] < X[2]$	1	$X[3] < X[1] < X[2]$	1
$X[2] < X[1] < X[3]$	0	$X[3] < X[2] < X[1]$	2

Среднее значение A при $n = 3$ равно $(0 + 1 + 0 + 1 + 1 + 2)/6 = 5/6$.

Очевидно, что можно считать $X[1], X[2], \ldots, X[n]$ числами $1, 2, \ldots, n$, расположенными в определенном порядке. Согласно нашим предположениям все $n!$ перестановок равновероятны. *Вероятность* того, что A имеет значение k, равна

$$p_{nk} = (\text{число перестановок } n \text{ объектов, для которых } A = k)/n!. \tag{1}$$

Для нашего примера (см. таблицу выше) $p_{30} = \frac{1}{3}$, $p_{31} = \frac{1}{2}$, $p_{32} = \frac{1}{6}$.

Среднее значение (или просто среднее) определяется, как обычно, по формуле

$$A_n = \sum_k k p_{nk}. \tag{2}$$

Дисперсия V_n определяется как среднее значение величины $(A - A_n)^2$, поэтому

$$\begin{aligned} V_n = \sum_k (k - A_n)^2 p_{nk} &= \sum_k k^2 p_{nk} - 2A_n \sum_k k p_{nk} + A_n^2 \sum_k p_{nk} \\ &= \sum_k k^2 p_{nk} - 2A_n A_n + A_n^2 = \sum_k k^2 p_{nk} - A_n^2. \end{aligned} \tag{3}$$

И наконец, *среднее квадратичное отклонение* σ_n определяется как $\sqrt{V_n}$.

Чтобы разъяснить смысл характеристики σ_n, заметим, что для всех $r \geq 1$ вероятность того, что значение A не попадает в промежуток $(A_n - r\sigma_n,\ A_n + r\sigma_n)$, не превышает $1/r^2$. Например, событие $|A - A_n| > 2\sigma_n$ происходит с вероятностью $< 1/4$. (*Доказательство.* Пусть p — требуемая вероятность*. Тогда если $p > 0$, то среднее величины $(A - A_n)^2$ больше, чем $p \cdot (r\sigma_n)^2 + (1 - p) \cdot 0$, т. е. $V_n > pr^2 V_n$.) Обычно это соотношение называют *неравенством Чебышева*, хотя на самом деле оно было впервые получено Ж. Бьенэме (J. Bienaymé) [*Comptes Rendus Acad. Sci. Paris* **37** (1853), 320–321].

Чтобы определить поведение A, найдем вероятности p_{nk}. Это легко сделать методом индукции. Согласно (1) нужно подсчитать число перестановок n элементов, для которых $A = k$. Обозначим это число через P_{nk}. Оно равно $P_{nk} = n! p_{nk}$.

Рассмотрим перестановки $x_1 x_2 \ldots x_n$ элементов $\{1, 2, \ldots, n\}$ (см. раздел 1.2.5). Если $x_1 = n$, то значение A *на единицу больше*, чем значение для перестановки $x_2 \ldots x_n$. Если же $x_1 \neq n$, то значение A является *точно таким же*, как для перестановки $x_2 \ldots x_n$. Следовательно, $P_{nk} = P_{(n-1)(k-1)} + (n - 1)P_{(n-1)k}$, что эквивалентно соотношению

$$p_{nk} = \frac{1}{n} p_{(n-1)(k-1)} + \frac{n-1}{n} p_{(n-1)k}. \tag{4}$$

---

* $p = P\{|A - A_n| > 2\sigma_n\}$. — *Прим. ред.*

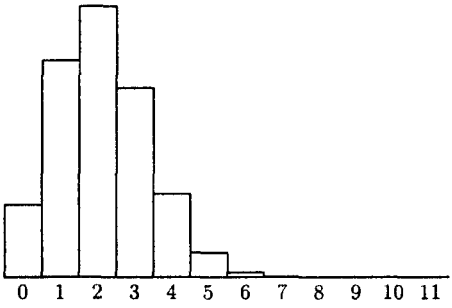


Рис. 10. Распределение вероятностей для шага M4 при $n=12$ Среднее равно 58301/27720, т е приближенно равно 2.10. Дисперсия приближенно равна 1 54

По этой формуле можно найти p_{nk}, если задать начальные условия:

$$p_{1k} = \delta_{0k}; \qquad p_{nk} = 0, \quad \text{если } k < 0 \tag{5}$$

Теперь можно исследовать величины p_{nk} с помощью производящих функций. Пусть

$$G_n(z) = p_{n0} + p_{n1}z + \cdots = \sum_k p_{nk} z^k. \tag{6}$$

Известно, что $A \le n-1$, поэтому $p_{nk} = 0$ для больших значений k Таким образом, функция $G_n(z)$ — это на самом деле многочлен, хотя для удобства она и представлена в виде бесконечного ряда

Из (5) получаем, что $G_1(z) = 1$, и согласно (4)

$$G_n(z) = \frac{z}{n} G_{n-1}(z) + \frac{n-1}{n} G_{n-1}(z) = \frac{z+n-1}{n} G_{n-1}(z). \tag{7}$$

(Читателю следует внимательно исследовать связь между соотношениями (4) и (7).) Теперь мы видим, что

$$\begin{aligned} G_n(z) = \frac{z+n-1}{n} G_{n-1}(z) &= \frac{z+n-1}{n} \frac{z+n-2}{n-1} G_{n-2}(z) = \cdots \\ &= \frac{1}{n!}(z+n-1)(z+n-2)\ldots(z+1) \\ &= \frac{1}{z+n}\binom{z+n}{n}. \end{aligned} \tag{8}$$

Таким образом, $G_n(z)$ — это, в сущности, биномиальный коэффициент!

Рассматриваемая здесь функция уже встречалась в предыдущем разделе (соотношение 1.2.9–(27)), где мы получили

$$G_n(z) = \frac{1}{n!} \sum_k \begin{bmatrix} n \\ k \end{bmatrix} z^{k-1}.$$

Следовательно, p_{nk} можно выразить с помощью чисел Стирлинга:

$$p_{nk} = \begin{bmatrix} n \\ k+1 \end{bmatrix} \Big/ n!. \tag{9}$$

На рис. 10 показаны приближенные значения p_{nk} при $n = 12$.

Теперь остается только подставить значение p_{nk} в (2) и (3) и получить нужную среднюю величину. Но это легче сказать, чем сделать. На самом деле очень редко удается явно определить вероятности p_{nk}. В большинстве задач будет известна производящая функция $G_n(z)$, но не сами значения вероятностей. Важно то, что *можно легко определить среднее и дисперсию по самой производящей функции.*

Чтобы понять, как это делается, предположим, что имеется производящая функция, коэффициенты которой представляют собой некоторые вероятности:

$$G(z) = p_0 + p_1 z + p_2 z^2 + \cdots.$$

Здесь p_k — вероятность того, что некоторая случайная величина принимает значение k. Вычислим величины*

$$\operatorname{mean}(G) = \sum_k k p_k, \quad \operatorname{var}(G) = \sum_k k^2 p_k - \big(\operatorname{mean}(G)\big)^2 \tag{10}$$

Это нетрудно сделать с помощью операции дифференцирования. Заметьте, что

$$G(1) = 1, \tag{11}$$

так как $G(1) = p_0 + p_1 + p_2 + \cdots$ — сумма всех возможных вероятностей. Аналогично поскольку $G'(z) = \sum_k k p_k z^{k-1}$, то

$$\operatorname{mean}(G) = \sum_k k p_k = G'(1). \tag{12}$$

И наконец, еще раз применив операцию дифференцирования, получим

$$\operatorname{var}(G) = G''(1) + G'(1) - G'(1)^2 \tag{13}$$

(см. упр. 2) В формулах (12) и (13) среднее и дисперсия выражены через производящую функцию.

В нашем случае нужно вычислить $G_n'(1) = A_n$ Из (7) имеем

$$G_n'(z) = \frac{1}{n} G_{n-1}(z) + \frac{z+n-1}{n} G_{n-1}'(z);$$

$$G_n'(1) = \frac{1}{n} + G_{n-1}'(1).$$

Учитывая начальное условие $G_1'(1) = 0$, находим

$$A_n = G_n'(1) = H_n - 1. \tag{14}$$

Это и есть среднее число выполнений шага М4. Для больших n оно приближенно равно $\ln n$ [*Замечание.* r-й момент $A + 1$, т. е. величина $\sum_k (k+1)^r p_{nk}$, есть $[z^n](1-z)^{-1} \sum_k \left\{ {r \atop k} \right\} \big(\ln \frac{1}{1-z}\big)^k$**, что приближенно равно $(\ln n)^r$ (см P B. M Roes, *CACM* **9** (1966), 342). Распределение вероятностей для величины A впервые было изучено Ф Г. Фостером (F. G Foster) и А. Стюартом (A Stuart), *J. Roy. Stat. Soc.* **B16** (1954), 1–22.]

Аналогично можно вычислить величину дисперсии V_n. Но сначала сформулируем теорему, позволяющую упростить нашу задачу.

* Поскольку между целочисленными случайными величинами и производящими функциями существует взаимно однозначное соответствие, ясно, что символы $\operatorname{mean}(G)$ и $\operatorname{var}(G)$ обозначают среднее и дисперсию случайной величины с производящей функцией G — *Прим. ред*

** $[z^n]$ — это коэффициент при z^n в разложении функции $f(z)$ в степенной ряд. — *Прим. ред.*

Теорема А. *Пусть G и H — две производящие функции, такие, что $G(1)=H(1)=1$, а величины* mean(G) *и* var(G) *определяются формулами* (12) *и* (13). *Тогда справедливы равенства*

$$mean(GH) = mean(G) + mean(H); \qquad var(GH) = var(G) + var(H) \tag{15}$$

Данная теорема говорит о том, что среднее (дисперсия) произведения производящих функций равно сумме средних (дисперсий) производящих функций*. Мы докажем ее несколько позже. ▮

Полагая $Q_n(z) = (z+n-1)/n$, имеем $Q_n'(1) = 1/n$, $Q_n''(1) = 0$. Отсюда

$$\text{mean}(Q_n) = \frac{1}{n}, \quad \text{var}(Q_n) = \frac{1}{n} - \frac{1}{n^2}.$$

И наконец, так как $G_n(z) = \prod_{k=2}^{n} Q_k(z)$, следовательно,

$$\text{mean}(G_n) = \sum_{k=2}^{n} \text{mean}(Q_k) = \sum_{k=2}^{n} \frac{1}{k} = H_n - 1,$$

$$\text{var}(G_n) = \sum_{k=2}^{n} \text{var}(Q_k) = \sum_{k=1}^{n} \left(\frac{1}{k} - \frac{1}{k^2}\right) = H_n - H_n^{(2)}$$

Подытожив полученные результаты, получим искомые статистические характеристики величины A:

$$A = \left(\min 0, \quad \text{ave } H_n - 1, \quad \max n - 1, \quad \text{dev } \sqrt{H_n - H_n^{(2)}}\right). \tag{16}$$

Такая форма записи, как в (16), будет использоваться для описания статистических характеристик вероятностных величин на протяжении всей книги.

Итак, мы закончили анализ алгоритма М; новой особенностью этого анализа явилось применение основ теории вероятностей. Для большинства приложений, рассматриваемых в данной книге, вполне достаточно элементарных сведений по теории вероятностей. Определения и простые методы вычисления среднего, дисперсии и среднего квадратичного отклонения, которые мы уже рассмотрели, позволят найти ответы на большинство поставленных вопросов. Более сложные алгоритмы помогут развить способность свободно пользоваться инструментами теории вероятностей.

Давайте рассмотрим несколько простых вероятностных задач, чтобы немного попрактиковаться в использовании этих методов. В связи с теорией вероятностей первое, что приходит в голову, — это задача о бросании монеты. Предположим, мы подбросили монету n раз и вероятность выпадения "орла" равна p. Сколько раз в среднем выпадет "орел"? Чему равно среднее квадратичное отклонение?

Рассмотрим случай несимметричной монеты, т. е. такой, для которой выпадения "орла" и "решки" не равновероятны. Таким образом, мы не считаем, что $p = \frac{1}{2}$. Это делает задачу более интересной; к тому же любая реальная монета несимметрична (иначе мы не могли бы отличить одну сторону от другой).

* Автор имеет в виду, что среднее (дисперсия) случайной величины, соответствующей производящей функции GH (сумме двух независимых случайных величин, соответствующих производящим функциям G и H), равно сумме средних (дисперсий) случайных величин, соответствующих производящим функциям G и H — *Прим. ред*

Теперь продолжим рассуждения. Пусть p_{nk} — вероятность того, что "орел" выпал k раз, и пусть $G_n(z)$ — соответствующая производящая функция Очевидно, что

$$p_{nk} = p\,p_{(n-1)(k-1)} + q\,p_{(n-1)k}, \tag{17}$$

где $q = 1 - p$ — вероятность выпадения 'решки". Как и раньше. из (17) получаем, что $G_n(z) = (q+pz)G_{n-1}(z)$, и в силу очевидного начального условия $G_1(z) = q+pz$ имеем

$$G_n(z) = (q + pz)^n \tag{18}$$

Отсюда по теореме A получаем

$$\operatorname{mean}(G_n) = n \operatorname{mean}(G_1) = pn;$$
$$\operatorname{var}(G_n) = n \operatorname{var}(G_1) = (p - p^2)n = pqn.$$

Таким образом для числа выпадений "орла" получаем следующие характеристики:

$$(\min 0. \quad \text{ave}\, pn, \quad \max n, \quad \text{dev}\, \sqrt{pqn}\,). \tag{19}$$

На рис. 11 показаны значения p_{nk} при $p = \frac{3}{5}$, $n = 12$. Если среднее квадратичное отклонение пропорционально $\sqrt{n}$ и разность между максимумом и минимумом пропорциональна n, можно считать, что ситуация "устойчива" относительно среднего.

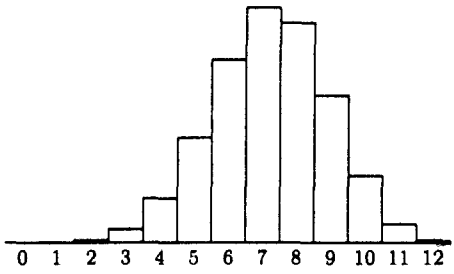


Рис. 11. Распределение вероятностей для задачи о бросании монеты: 12 независимых испытаний с вероятностью успеха, равной 3/5 в каждом испытании.

Давайте решим еще одну простую задачу. Предположим, при выполнении некоторого процесса существуют *равновероятные* возможности получения значений $1, 2, \ldots, n$. Производящая функция в этом случае имеет вид

$$G(z) = \frac{1}{n}z + \frac{1}{n}z^2 + \cdots + \frac{1}{n}z^n = \frac{1}{n}\,\frac{z^{n+1} - z}{z - 1}. \tag{20}$$

После достаточно трудоемких выкладок находим, что

$$G'(z) = \frac{nz^{n+1} - (n+1)z^n + 1}{n(z-1)^2};$$
$$G''(z) = \frac{n(n-1)z^{n+1} - 2(n+1)(n-1)z^n + n(n+1)z^{n-1} - 2}{n(z-1)^3}.$$

Теперь, чтобы найти среднее и дисперсию, нужно вычислить $G'(1)$ и $G''(1)$; но если просто подставить значение $z = 1$ в формулы, то получится выражение вида $0/0$.

Поэтому возникает необходимость найти предел при z, стремящемся к единице, но это нетривиальная задача.

К счастью, существует гораздо более простой способ дальнейшего решения задачи. По теореме Тейлора (Taylor) имеем

$$G(1+z) = G(1) + G'(1)z + \frac{G''(1)}{2!}z^2 + \cdots. \tag{21}$$

Таким образом, нужно просто заменить в формуле (20) z на $z+1$ и найти коэффициенты:

$$G(1+z) = \frac{1}{n}\frac{(1+z)^{n+1}-1-z}{z} = 1 + \frac{n+1}{2}z + \frac{(n+1)(n-1)}{6}z^2 + \cdots.$$

Следовательно, $G'(1) = \frac{1}{2}(n+1)$, $G''(1) = \frac{1}{3}(n+1)(n-1)$. В результате получаем следующие статистические характеристики для равномерного распределения:

$$\left(\min 1, \quad \operatorname{ave} \frac{n+1}{2}, \quad \max n, \quad \operatorname{dev} \sqrt{\frac{(n+1)(n-1)}{12}}\right). \tag{22}$$

В таком случае среднее квадратичное отклонение, равное приблизительно $0.289n$, указывает на явно *неустойчивую* ситуацию.

В завершение этого раздела докажем теорему А и обоснуем введенные нами понятия с точки зрения классической теории вероятностей. Пусть X — случайная величина, которая принимает только неотрицательные целые значения и $X = k$ с вероятностью p_k для любого k. Тогда $G(z) = p_0 + p_1 z + p_2 z^2 + \cdots$ называется *производящей функцией вероятностей* для X, а $G(e^{it}) = p_0 + p_1 e^{it} + p_2 e^{2it} + \cdots$ — *характеристической функцией* этого распределения. Распределение, соответствующее произведению двух таких производящих функций, называется *сверткой* двух соответствующих распределений и представляет собой распределение суммы двух независимых случайных величин, имеющих распределения, соответствующие перемножаемым производящим функциям.

Среднее значение случайной величины X часто называют ее *математическим ожиданием* и обозначают через $\mathrm{E}\,X$. Тогда дисперсия случайной величины X равна $\mathrm{E}\,X^2 - (\mathrm{E}\,X)^2$. В этих обозначениях производящая функция, соответствующая распределению случайной величины X, имеет вид $G(z) = \mathrm{E}\,z^X$, т. е. представляет собой математическое ожидание случайной величины z^X, если X принимает только целые неотрицательные значения. Аналогично, если X — утверждение, которое либо истинно, либо ложно, вероятность того, что X истинно, равна $\Pr(X) = \mathrm{E}[X]$ согласно обозначению Айверсона (см. 1.2.3–(16)).

Среднее и дисперсия — это просто два так называемых *семиинварианта* или *кумулянта*, введенных Тиеле (Thiele) в 1903 году. Семиинварианты $\kappa_1, \kappa_2, \kappa_3, \ldots$ определяются правилом

$$\frac{\kappa_1 t}{1!} + \frac{\kappa_2 t^2}{2!} + \frac{\kappa_3 t^3}{3!} + \cdots = \ln G(e^t). \tag{23}$$

Имеем

$$\kappa_n = \frac{d^n}{dt^n} \ln G(e^t)\Big|_{t=0};$$

в частности,

$$\kappa_1 = \frac{e^t G'(e^t)}{G(e^t)}\Big|_{t=0} = G'(1),$$

так как $G(1) = \sum_k p_k = 1$, и

$$\kappa_2 = \frac{e^{2t}G''(e^t)}{G(e^t)} + \frac{e^t G'(e^t)}{G(e^t)} - \frac{e^{2t}G'(e^t)^2}{G(e^t)^2}\Big|_{t=0} = G''(1) + G'(1) - G'(1)^2.$$

Поскольку семиинварианты определяются с помощью *логарифма* производящей функции, теорема А очевидна; фактически ее можно обобщить, распространив на все семиинварианты.

Нормальное распределение — это такое распределение, для которого все семиинварианты, за исключением среднего и дисперсии, равны нулю Для нормального распределения можно значительно улучшить неравенство Чебышева. вероятность того, что разница между значением нормально распределенной случайной величины и средним меньше среднего квадратичного отклонения, равна

$$\frac{1}{\sqrt{2\pi}} \int_{-1}^{+1} e^{-t^2/2}\, dt,$$

т. е. приблизительно 0.68268949213709. Вероятность, что эта разница меньше удвоенного среднего квадратичного отклонения, приблизительно равна 0.95449973610364, и вероятность того, что она меньше, чем утроенное среднее квадратичное отклонение, примерно равна 0 99730020393674. Распределения, заданные соотношениями (8) и (18), *приблизительно* нормальны при больших значениях n (см. упр. 13 и 14).

Часто возникает необходимость убедиться в том, что вероятность большого отклонения случайной величины от ее среднего достаточно мала. Удобные оценки таких вероятностей дают две чрезвычайно простые, но очень важные формулы, которые называются *неравенствами для хвостов распределений*. Если $G(z)$ — вероятностная производящая функция случайной величины X, то

$$\Pr(X \le r) \le x^{-r} G(x) \qquad \text{для } 0 < x \le 1; \tag{24}$$

$$\Pr(X \ge r) \le x^{-r} G(x) \qquad \text{для } x \ge 1. \tag{25}$$

Доказать эти формулы несложно. Если $G(z) = p_0 + p_1 z + p_2 z^2 + \cdots$, то

$$\Pr(X \le r) = p_0 + p_1 + \cdots + p_{\lfloor r \rfloor} \le x^{-r} p_0 + x^{1-r} p_1 + \cdots + x^{\lfloor r \rfloor - r} p_{\lfloor r \rfloor} \le x^{-r} G(x)$$

при $0 < x \le 1$ и

$$\Pr(X \ge r) = p_{\lceil r \rceil} + p_{\lceil r \rceil + 1} + \cdots \le x^{\lceil r \rceil - r} p_{\lceil r \rceil} + x^{\lceil r \rceil + 1 - r} p_{\lceil r \rceil + 1} + \cdots \le x^{-r} G(x)$$

при $x \ge 1$. Выбирая такие значения x, которые минимизируют или приближенно минимизируют правые части неравенств (24) и (25), можно получить оценки сверху, достаточно близкие к истинным значениям вероятностей слева.

В упр. 21–23 неравенства (24) и (25) проиллюстрированы для нескольких важных случаев. Эти неравенства являются частными случаями более общего закона, на который указал А. Н. Колмогоров (A. N. Kolmogorov) в книге *Grundbegriffe der Wahrscheinlichkeitsrechnung* (Springer, 1933): если $f(t) \ge s > 0$ для всех $t \ge r$, то

$\Pr(X \geq r) \leq s^{-1}\,\mathrm{E}\,f(X)$ при условии, что существует $\mathrm{E}\,f(X)$ Неравенство (25) можно получить для $f(t) = x^t$ и $s = x^r$

УПРАЖНЕНИЯ

1. [*10*] Найдите значение p_{n0} из соотношений (4) и (5) и дайте интерпретацию этой вероятности в свете алгоритма M

2. [*HM16*] Выведите (13) из (10).

3. [*M15*] Чему будут равны минимум, максимум, среднее и среднее квадратичное отклонение для числа выполнений шага M4, если для нахождения максимума среди 1 000 случайно упорядоченных различных величин воспользоваться алгоритмом M? (Дайте ответ в виде десятичных дробей.)

4. [*M10*] Дайте в явном виде формулу для p_{nk} из задачи о бросании монеты (см. соотношение (17))

5. [*M13*] Чему равны среднее и среднее квадратичное отклонение для распределения, показанного на рис 11?

6. [*HM27*] Мы вычислили среднее и дисперсию для важных распределений вероятностей (8), (18), (20). Чему равен *третий* семиинвариант, κ_3, в каждом из этих случаев?

▶ **7.** [*M27*] Анализируя алгоритм M, мы предполагали, что все $X[k]$ были различны. Теперь сделаем более слабое предположение, а именно — что среди $X[1], X[2], \ldots, X[n]$ содержится ровно m различных значений; в других отношениях эти величины случайны. Каким будет распределение вероятностей для A в этом случае?

▶ **8.** [*M20*] Предположим, что каждое $X[k]$ выбирается наугад из множества M различных элементов, так что все M^n возможных вариантов выбора элементов $X[1], X[2], \ldots, X[n]$ считаются равновероятными. Чему равна вероятность того, что все $X[k]$ будут различны?

9. [*M25*] Обобщите результат предыдущего упражнения и найдите формулу для вероятности того, что среди $X[k]$ окажется ровно m различных величин. Выразите эту вероятность с помощью чисел Стирлинга

10. [*M20*] С помощью результатов трех предыдущих упражнений выведите формулу для вероятности того, что $A = k$, при условии, что каждое $X[k]$ выбирается наугад из M-элементного множества.

▶ **11.** [*M15*] Что произойдет с семиинвариантами распределения, если заменить функцию $G(z)$ функцией $F(z) = z^n G(z)$?

12. [*HM21*] Если $G(z) = p_0 + p_1 z + p_2 z^2 + \cdots$ соответствует некоторому распределению вероятностей, то величины $M_n = \sum_k k^n p_k$ и $m_n = \sum_k (k - M_1)^n p_k$ называются n-м моментом и n-м центральным моментом соответственно. Покажите, что $G(e^t) = 1 + M_1 t + M_2 t^2/2! + \cdots$ С помощью формулы Арбогаста (см. упр. 1.2.5–21) докажите, что

$$\kappa_n = \sum_{\substack{k_1, k_2, \ldots, k_n \geq 0 \\ k_1 + 2k_2 + \cdots + nk_n = n}} \frac{(-1)^{k_1+k_2+\cdots+k_n-1}\, n!\, (k_1 + k_2 + \cdots + k_n - 1)!}{k_1!\, 1!^{k_1}\, k_2!\, 2!^{k_2} \ldots k_n!\, n!^{k_n}} M_1^{k_1} M_2^{k_2} \ldots M_n^{k_n}.$$

В частности, $\kappa_1 = M_1$, $\kappa_2 = M_2 - M_1^2$ (как мы уже знаем), $\kappa_3 = M_3 - 3M_1M_2 + 2M_1^3$ и $\kappa_4 = M_4 - 4M_1M_3 + 12M_1^2M_2 - 3M_2^2 - 6M_1^4$ Найдите аналогичное выражение для κ_n через центральные моменты $m_2, m_3, \ldots$, где $n \geq 2$

13. [*HM38*] Говорят, что последовательность распределений вероятностей, соответствующих производящим функциям $G_n(z)$ со средними μ_n и средними квадратичными отклонениями σ_n, *стремится к нормальному распределению*, если

$$\lim_{n\to\infty} e^{-it\mu_n/\sigma_n} G_n(e^{it/\sigma_n}) = e^{-t^2/2}$$

для всех действительных значений t. Пусть $G_n(z)$ задается формулой (8). Покажите, что распределение, соответствующее $G_n(z)$, стремится к нормальному распределению.

Замечание. Можно показать, что данное здесь определение стремления к нормальному распределению эквивалентно следующей формуле:

$$\lim_{n\to\infty} \text{вероятность}\left(\frac{X_n - \mu_n}{\sigma_n} \le x\right) = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{x} e^{-t^2/2}\,dt,$$

где X_n — случайная величина, распределение вероятностей которой задается с помощью $G_n(z)$. Это частный случай важной "теоремы непрерывности" П. Леви (P. Lévy), которая является одним из основных результатов математической теории вероятностей. Доказательство теоремы Леви выходит за рамки данной книги, хотя оно не такое уж сложное [см., например, книгу Б. В. Гнеденко и А. Н. Колмогорова, *Предельные распределения для сумм независимых случайных величин* (М.: Гостехиздат, 1949)].

14. [*HM30*] (А. де Муавр.) Пользуясь обозначениями из предыдущего упражнения, покажите, что биномиальное распределение $G_n(z)$, заданное формулой (18), стремится к нормальному распределению.

15. [*HM23*] Если вероятность того, что некоторая случайная величина принимает значение k, равна $e^{-\mu}(\mu^k/k!)$*, то говорят, что она имеет *распределение Пуассона со средним* μ.

a) Найдите производящую функцию для этого распределения вероятностей.

b) Найдите значения семиинвариантов.

c) Покажите, что при $n \to \infty$ распределение Пуассона со средним np стремится к нормальному распределению в смысле упр. 13.

16. [*M25*] Пусть распределение случайной величины X является *смесью* распределений, порожденных функциями $g_1(z)$, $g_2(z)$, ..., $g_r(z)$, в том смысле, что распределение X с вероятностью p_k совпадает с распределением случайной величины, соответствующей производящей функции $g_k(z)$, где $p_1 + p_2 + \cdots + p_r = 1$. Найдите производящую функцию для X. Выразите среднее и дисперсию X через средние и дисперсии g_1, g_2, ..., g_r.

▶ **17.** [*M27*] Пусть $f(z)$ и $g(z)$ — производящие функции, которые соответствуют некоторым вероятностным распределениям.

a) Покажите, что $h(z) = g(f(z))$ — тоже производящая функция, соответствующая некоторому вероятностному распределению.

b) Дайте интерпретацию $h(z)$ в терминах $f(z)$ и $g(z)$. (Каков *смысл* вероятностей, заданных коэффициентами разложения $h(z)$?)

c) Выразите среднее и дисперсию h через средние и дисперсии f и g.

18. [*M28*] Предположим, что величины, которые мы обозначили через $X[1], X[2], \ldots, X[n]$ в описании алгоритма M, содержат ровно k_1 единиц, k_2 двоек, ..., k_n чисел n, расположенных в случайном порядке. (Здесь

$$k_1 + k_2 + \cdots + k_n = n.$$

В тексте предполагалось, что $k_1 = k_2 = \cdots = k_n = 1$.) Покажите, что в этой более общей ситуации производящая функция (8) будет иметь вид

$$\left(\frac{k_{n-1}z + k_n}{k_{n-1} + k_n}\right)\left(\frac{k_{n-2}z + k_{n-1} + k_n}{k_{n-2} + k_{n-1} + k_n}\right)\cdots\left(\frac{k_1 z + k_2 + \cdots + k_n}{k_1 + k_2 + \cdots + k_n}\right),$$

если принять, что $0/0 = 1$.

* $\mu > 0$ — *Прим. ред.*

19. [*M21*] Если $a_k > a_j$ для $1 \le j < k$, будем говорить, что a_k — это *максимум слева направо* последовательности $a_1\, a_2\, \ldots\, a_n$. Предположим, что $a_1\, a_2 \ldots a_n$ — это перестановка чисел $\{1, 2, \ldots, n\}$, и $b_1\, b_2 \ldots b_n$ — обратная перестановка, так что $a_k = l$ тогда и только тогда, когда $b_l = k$. Покажите, что a_k является максимумом слева направо последовательности $a_1\, a_2 \ldots a_n$ тогда и только тогда, когда k — это максимум справа налево последовательности $b_1\, b_2 \ldots b_n$.

▸ **20.** [*M22*] Предположим, нужно вычислить $\max\{|a_1 - b_1|, |a_2 - b_2|, \ldots, |a_n - b_n|\}$, где $b_1 \le b_2 \le \cdots \le b_n$. Покажите, что достаточно вычислить $\max\{m_L, m_R\}$, где

$$m_L = \max\{a_k - b_k \mid a_k \text{ — максимум справа налево последовательности } a_1,\ a_2 \ldots a_n\},$$
$$m_R = \max\{b_k - a_k \mid a_k \text{ — минимум справа налево последовательности } a_1,\ a_2 \ldots a_n\}.$$

[Таким образом, если a_k расположены в случайном порядке, то число таких k, для которых необходимо выполнить вычитание, приблизительно равно $2 \ln n$.]

▸ **21.** [*HM21*] Пусть монета бросается наудачу n раз и X — число выпадений "орла" в этой серии испытаний. Распределению вероятностей для X соответствует производящая функция (18). Воспользуйтесь (25) для доказательства того, что

$$\Pr(X \ge n(p + \epsilon)) \le e^{-\epsilon^2 n/(2q)},$$

где $\epsilon \ge 0$, и получите аналогичную оценку для $\Pr(X \le n(p - \epsilon))$*.

▸ **22.** [*HM22*] Предположим, что X имеет производящую функцию $(q_1 + p_1 z)(q_2 + p_2 z) \ldots (q_n + p_n z)$, где $p_k + q_k = 1$ для $1 \le k \le n$. Пусть $\mu = \mathrm{E}\, X = p_1 + p_2 + \cdots + p_n$. (a) Докажите, что

$$\Pr(X \le \mu r) \le (r^{-r} e^{r-1})^{\mu}, \text{ когда } 0 < r \le 1,$$
$$\Pr(X \ge \mu r) \le (r^{-r} e^{r-1})^{\mu}, \text{ когда } r \ge 1.$$

(b) Выразите правые части этих оценок в более удобном виде, когда $r \approx 1$ (c) Покажите, что если r достаточно большое, то имеем $\Pr(X \ge \mu r) \le 2^{-\mu r}$.

23. [*HM23*] Укажите неравенства для хвостов распределений для случайной величины, имеющей *отрицательное биномиальное распределение*, т. е. распределение, которому соответствует производящая функция $(q - pz)^{-n}$, где $q = p + 1$.

*1.2.11. Асимптотические представления

Во многих случаях для того, чтобы сравнить одну величину с другой, достаточно знать не точные, а приближенные их значения. Например, формула Стирлинга для $n!$ — это удобное приближение подобного типа для больших n; мы пользовались также приближением $H_n \approx \ln n + \gamma$. При выводе подобных *асимптотических формул* обычно используются методы высшей математики, но в следующих разделах для получения нужных результатов мы не будем выходить за рамки элементарной математики.

***1.2.11.1. Символ O.** Поль Бахман (Paul Bachmann) в книге *Analytische Zahlentheorie* (1894 г.) ввел очень удобное обозначение для использования в приближенных формулах. Это символ O, который позволяет заменить знак "$\approx$" знаком "$=$" и количественно выразить степень точности, например

$$H_n = \ln n + \gamma + O\Big(\frac{1}{n}\Big). \tag{1}$$

---

* Здесь p — вероятность того, что выпадет "орел", а $q = 1 - p$. — *Прим. ред.*

(Читается эта запись так: "H_n равно натуральному логарифму от n плюс постоянная Эйлера плюс о большое от единицы на n".)

Вообще говоря, каждый раз, когда $f(n)$ является функцией от положительного целого n, можно использовать запись $O(f(n))$; она обозначает *величину, точное значение которой неизвестно*, и известно только, что ее значение не слишком велико*. Запись $O(f(n))$ всегда обозначает следующее: существуют положительные константы M и n_0, такие, что величина x_n, представленная в виде $O(f(n))$, удовлетворяет условию $|x_n| \le M\,|f(n)|$ для всех целых $n \ge n_0$. Мы не можем сказать, *каковы* на самом деле эти константы M и n_0, так как в каждом случае они зависят от соотношения, в котором использован символ O.

Например, соотношение (1) означает, что $|H_n - \ln n - \gamma| \le M/n$, когда $n \ge n_0$. Хотя значения констант M и n_0 не указаны, мы можем быть уверены, что для достаточно большого n величина $O(1/n)$ будет сколь угодно малой.

Рассмотрим еще несколько примеров. Мы знаем, что

$$1^2 + 2^2 + \quad + n^2 = \tfrac{1}{3}n(n+\tfrac{1}{2})(n+1) = \tfrac{1}{3}n^3 + \tfrac{1}{2}n^2 + \tfrac{1}{6}n.$$

Отсюда следует, что

$$1^2 + 2^2 + \cdots + n^2 = O(n^4), \tag{2}$$

$$1^2 + 2^2 + \cdots + n^2 = O(n^3), \tag{3}$$

$$1^2 + 2^2 + \cdots + n^2 = \tfrac{1}{3}n^3 + O(n^2). \tag{4}$$

Соотношение (2) достаточно грубое, хотя и правильное. Соотношение (3) является более сильным, а (4) — еще сильнее. Чтобы подтвердить правильность этих соотношений, докажем, что *если* $P(n) = a_0 + a_1 n + \cdots + a_m n^m$ *— произвольный многочлен степени, меньшей или равной* m, *то* $P(n) = O(n^m)$. Это вытекает из следующих оценок:

$$\begin{aligned} |P(n)| \le |a_0| + |a_1|\,n + \cdots + |a_m|\,n^m &= \big(|a_0|/n^m + |a_1|/n^{m-1} + \cdots + |a_m|\big)n^m \\ &\le \big(|a_0| + |a_1| + \cdots + |a_m|\big)n^m, \end{aligned}$$

где $n \ge 1$. Поэтому можно взять $M = |a_0| + |a_1| + \cdots + |a_m|$ и $n_0 = 1$. Можно было бы взять также, скажем, $M = |a_0|/2^m + |a_1|/2^{m-1} + \cdots + |a_m|$ и $n_0 = 2$.

Символ O очень полезен в работе с приближенными формулами, так как позволяет кратко описать суть дела, опустив ненужные детали. Более того, с символами O можно выполнять хорошо известные алгебраические операции, хотя нужно иметь в виду некоторые особенности. Самый важный момент заключается в *односторонности равенств*: мы пишем $\frac{1}{2}n^2 + n = O(n^2)$, но ни в коем случае *не* $O(n^2) = \frac{1}{2}n^2 + n$. (В противном случае, так как $\frac{1}{4}n^2 = O(n^2)$, можно прийти к полному абсурду, получив равенство $\frac{1}{4}n^2 = \frac{1}{2}n^2 + n$.) Мы всегда следуем соглашению, что *правая часть равенства несет не больше информации, чем левая*; правая часть — это "огрубление" левой.

Соглашение по поводу использования знака "=" можно более точно сформулировать следующим образом: формулы, содержащие запись $O(f(n))$, можно рассматривать как множества функций от n. Запись $O(f(n))$ обозначает множество всех

* По сравнению с $f(n)$ — *Прим. ред.*

функций g от целых чисел, для которых существуют константы M и n_0, такие, что $|g(n)| \le M\,|f(n)|$ для всех целых $n \ge n_0$. Если S и T — множества функций, то $S+T$ обозначает множество $\{g+h \mid g \in S \text{ и } h \in T\}$; аналогично определяются множества $S+c$, $S-T$, $S\cdot T$, $\log S$ и т. д. Если $\alpha(n)$ и $\beta(n)$ — формулы, содержащие символ O, то запись $\alpha(n) = \beta(n)$ означает, что множество функций, относящихся к классу $\alpha(n)$, *содержится в* множестве функций, относящихся к классу $\beta(n)$.

Следовательно, большинство привычных операций можно выполнять, пользуясь знаком "=": если $\alpha(n) = \beta(n)$ и $\beta(n) = \gamma(n)$, то $\alpha(n) = \gamma(n)$. Кроме того, если $\alpha(n) = \beta(n)$ и если $\delta(n)$ — формула, полученная в результате подстановки $\beta(n)$ вместо некоторых $\alpha(n)$ в формуле $\gamma(n)$, то $\gamma(n) = \delta(n)$. Из этих двух утверждений следует, например, что если $g(x_1, x_2, \ldots, x_m)$ — любая действительная функция и если $\alpha_k(n) = \beta_k(n)$ для $1 \le k \le m$, то $g\big(\alpha_1(n), \alpha_2(n), \ldots, \alpha_m(n)\big) = g\big(\beta_1(n), \beta_2(n), \ldots, \beta_m(n)\big)$.

Вот некоторые простые операции, которые можно выполнять с символом O:

$$f(n) = O\big(f(n)\big), \tag{5}$$

$$c \cdot O\big(f(n)\big) = O\big(f(n)\big), \qquad \text{если } c \text{ — константа}, \tag{6}$$

$$O\big(f(n)\big) + O\big(f(n)\big) = O\big(f(n)\big), \tag{7}$$

$$O\big(O(f(n))\big) = O\big(f(n)\big), \tag{8}$$

$$O\big(f(n)\big)O\big(g(n)\big) = O\big(f(n)g(n)\big), \tag{9}$$

$$O\big(f(n)g(n)\big) = f(n)O\big(g(n)\big). \tag{10}$$

Символ O также часто используется с функциями комплексного переменного z в окрестности точки $z = 0$. Через $O\big(f(z)\big)$ мы обозначаем любую величину $g(z)$, такую, что $|g(z)| \le M\,|f(z)|$ при $|z| < r$. (Как и прежде, M и r — это некоторые неопределенные константы, хотя мы могли бы определить их, если бы захотели.) Применительно к O-записи всегда должны указываться используемая переменная и область ее изменения. Если у нас переменная n, то неявно предполагается, что в записи $O\big(f(n)\big)$ подразумеваются функции от большого целого n. Если же используется переменная z, то предполагается, что $O\big(f(z)\big)$ относится к функциям малого комплексного числа z.

Предположим, что $g(z)$ — это функция, заданная бесконечным степенным рядом

$$g(z) = \sum_{k \ge 0} a_k z^k,$$

сходящимся в точке $z = z_0$. Тогда сумма абсолютных значений $\sum_{k\ge 0} |a_k z^k|$ сходится также при $|z| < |z_0|$. Если $z_0 \ne 0$, то можно записать

$$g(z) = a_0 + a_1 z + \cdots + a_m z^m + O(z^{m+1}) \tag{11}$$

Так как $g(z) = a_0 + a_1 z + \cdots + a_m z^m + z^{m+1}(a_{m+1} + a_{m+2} z + \cdots)$, нужно только показать, что величина в скобках ограничена при $|z| \le r$, где r — некоторое положительное число. Действительно, величина $|a_{m+1}| + |a_{m+2}|\,r + |a_{m+3}|\,r^2 + \cdots$ является ее верхней гранью при $|z| \le r < |z_0|$.

Рассмотрим несколько примеров. Производящие функции, приведенные в разделе 1.2.9, для всех неотрицательных целых m дают важные асимптотические формулы для достаточно малых z:

$$e^z = 1 + z + \frac{1}{2!}z^2 + \cdots + \frac{1}{m!}z^m + O(z^{m+1}), \tag{12}$$

$$\ln(1+z) = z - \frac{1}{2}z^2 + \cdots + \frac{(-1)^{m+1}}{m}z^m + O(z^{m+1}), \tag{13}$$

$$(1+z)^\alpha = 1 + \alpha z + \binom{\alpha}{2}z^2 + \cdots + \binom{\alpha}{m}z^m + O(z^{m+1}), \tag{14}$$

$$\frac{1}{1-z}\ln\frac{1}{1-z} = z + H_2 z^2 + \cdots + H_m z^m + O(z^{m+1}). \tag{15}$$

Важно отметить, что константы M и r при каждом конкретном O зависят одна от другой. Например, очевидно, что функция $e^z = O(1)$ при $|z| \le r$ для любого фиксированного r, так как $|e^z| \le e^{|z|}$; но не существует константы M, такой, что $|e^z| \le M$ для всех z Поэтому по мере увеличения r нам придется брать все большее и большее значение M.

Иногда асимптотическая формула справедлива, хотя не существует соответствующего сходящегося бесконечного ряда Например, основные формулы

$$n^{\bar r} = \sum_{k=0}^{m} \begin{bmatrix} r \\ r-k \end{bmatrix} n^{r-k} + O(n^{r-m-1}), \tag{16}$$

$$n^{\underline r} = \sum_{k=0}^{m} (-1)^k \begin{bmatrix} r \\ r-k \end{bmatrix} n^{r-k} + O(n^{r-m-1}), \tag{17}$$

выражающие факториальные степени через обычные степени, асимптотически справедливы для любого действительного r и любого фиксированного целого $m \ge 0$, в то время как сумма

$$\sum_{k=0}^{\infty} \begin{bmatrix} 1/2 \\ 1/2-k \end{bmatrix} n^{1/2-k}$$

расходится для всех n (см. упр. 12). Разумеется, если r — неотрицательное целое, то $n^{\bar r}$ и $n^{\underline r}$ являются просто многочленами степени r и (17) — это, в сущности, то же, что и 1.2.6 (44) Если r — неотрицательное целое и $|n| > |r|$, то бесконечная сумма $\sum_{k=0}^{\infty}\left[\begin{smallmatrix} r \\ r-k \end{smallmatrix}\right] n^{r-k}$ сходится к $n^{\bar r} = 1/(n-1)^{\underline{-r}}$. Эту сумму можно записать также в более естественном виде, $\sum_{k=0}^{\infty}\left\{\begin{smallmatrix} k-r \\ -r \end{smallmatrix}\right\} n^{r-k}$, воспользовавшись соотношением 1.2.6-(58).

Приведем один простой пример для иллюстрации введенных понятий. Рассмотрим величину $\sqrt[n]{n}$. По мере увеличения n последовательность корней n-й степени из фиксированного числа будет убывать, но совсем не очевидно, убывающей или возрастающей будет последовательность $\sqrt[n]{n}$ Оказывается, что $\sqrt[n]{n}$ убывает и стремится к единице. Теперь давайте рассмотрим несколько более сложную величину $n(\sqrt[n]{n} - 1)$. Здесь $(\sqrt[n]{n} - 1)$ убывает при увеличении n. А как будет себя вести $n(\sqrt[n]{n} - 1)$?

Эта задача легко решается с помощью приведенных выше формул. Имеем

$$\sqrt[n]{n} = e^{\ln n/n} = 1 + (\ln n/n) + O\big((\ln n/n)^2\big), \tag{18}$$

так как $\ln n/n \to 0$ при $n \to \infty$ (см. упр. 8 и 11). Соотношение (18) доказывает утверждение о том, что $\sqrt[n]{n} \to 1$. Более того, из него следует, что

$$n(\sqrt[n]{n} - 1) = n\big(\ln n/n + O((\ln n/n)^2)\big) = \ln n + O\big((\ln n)^2/n\big). \tag{19}$$

Другими словами, $n(\sqrt[n]{n} - 1)$ приближенно равно $\ln n$; эти величины отличаются на величину $O\big((\ln n)^2/n\big)$, которая стремится к нулю при n, стремящемся к бесконечности.

Многие часто неправильно пользуются записями с символом O, считая, что они дают точный порядок роста, т. е. определяют как верхнюю, так и нижнюю грани. Например, алгоритм сортировки n чисел можно назвать неэффективным, "потому что время его выполнения составляет $O(n^2)$". Но из этого никак не следует, что время выполнения алгоритма не составляет также $O(n)$. Для нижних граней существует другая запись, с символом "большая омега". Утверждение

$$g(n) = \Omega\big(f(n)\big) \tag{20}$$

означает, что существуют положительные константы L и n_0, такие, что

$$\big|g(n)\big| \geq L\big|f(n)\big| \quad \text{для всех } n \geq n_0.$$

Эта запись позволяет сделать правильный вывод о том, что алгоритм сортировки, время выполнения которого равно $\Omega(n^2)$, будет не таким эффективным, как алгоритм, время выполнения которого равно $O(n \log n)$ (для достаточно больших n). Но, не зная констант, подразумеваемых записями с символами O и Ω, мы ничего не можем сказать о том, насколько большим должно быть n, чтобы метод $O(n \log n)$ начал выигрывать в эффективности.

И наконец, чтобы точно указать порядок роста, не давая при этом точных значений констант, можно воспользоваться записью с символом "большая тета":

$$g(n) = \Theta\big(f(n)\big) \qquad \Longleftrightarrow \qquad g(n) = O\big(f(n)\big) \text{ и } g(n) = \Omega\big(f(n)\big). \tag{21}$$

УПРАЖНЕНИЯ

1. [*HM01*] Чему равен $\lim_{n\to\infty} O(n^{-1/3})$?

▶ **2.** [*M10*] М-р Далл*, применяя "очевидную" формулу $O(f(n)) - O(f(n)) = 0$, получил удивительные результаты. В чем была его ошибка, и как должна выглядеть правая часть "очевидной" формулы?

3. [*M15*] Умножьте $(\ln n + \gamma + O(1/n))$ на $(n + O(\sqrt{n}))$ и представьте результат с помощью символа O.

▶ **4.** [*M15*] Дайте асимптотическое разложение $n(\sqrt[n]{a} - 1)$, где $a > 0$, с точностью до членов порядка $O(1/n^3)$.

5. [*M20*] Докажите или опровергните следующее: $O(f(n) + g(n)) = f(n) + O(g(n))$, если $f(n)$ и $g(n)$ положительны для всех n. (Ср. с формулой (10).)

▶ **6.** [*M20*] Где ошибка в следующем рассуждении? "Так как $n = O(n)$ и $2n = O(n)$, ..., то

$$\sum_{k=1}^{n} kn = \sum_{k=1}^{n} O(n) = O(n^2)\text{ "}$$

---

* В оригинале — B. C. Dull. От англ. "dull" ("тупица"). — *Прим. перев.*

7. [*HM15*] Докажите, что для произвольного целого m нельзя найти такое M, чтобы для сколь угодно больших значений x выполнялось неравенство $e^x \le Mx^m$.

8. [*HM20*] Докажите, что $(\ln n)^m/n \to 0$ при $n \to \infty$.

9. [*HM20*] Покажите, что $e^{O(z^m)} = 1 + O(z^m)$ для всех фиксированных $m \ge 0$.

10. [*HM22*] Сформулируйте утверждение, аналогичное утверждению из упр. 9, относительно $\ln(1 + O(z^m))$.

▸ **11.** [*M11*] Объясните, почему верна формула (18).

12. [*HM25*] Докажите, что $\left[{1/2 \atop 1/2-k}\right] n^{-k}$ не стремится к нулю при $k \to \infty$ для любого целого n. Воспользуйтесь тем, что $\left[{1/2 \atop 1/2-k}\right] = (-\frac{1}{2})^k [z^k] \left(ze^z/(e^z-1)\right)^{1/2}$.

▸ **13.** [*M10*] Докажите или опровергните следующее: $g(n) = \Omega(f(n))$ тогда и только тогда, когда $f(n) = O(g(n))$.

***1.2.11.2. Формула суммирования Эйлера.** Одним из лучших методов получения приближенных значений сумм является метод, предложенный Леонардом Эйлером. Он состоит в том, чтобы аппроксимировать конечную сумму интегралом, и во многих случаях позволяет получать приближения с любой степенью точности. [*Commentarii Academiæ Scientiarum Petropolitanæ* **6** (1732), 68–97.]

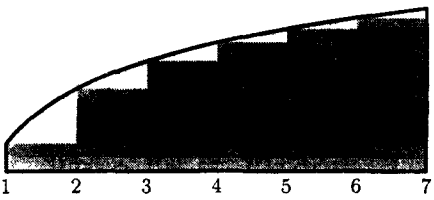


Рис. 12. Сравнение суммы с интегралом.

На рис. 12 сравниваются $\int_1^n f(x)\,dx$ и $\sum_{k=1}^{n-1} f(k)$ при $n = 7$. Предполагая, что $f(x)$ — дифференцируемая функция, с помощью метода Эйлера можно получить удобную формулу для разности между интегралом и суммой.

Для удобства введем следующее обозначение:

$$\{x\} = x \bmod 1 = x - \lfloor x \rfloor. \tag{1}$$

Начнем выкладки со следующего тождества:

$$\begin{aligned}\int_k^{k+1} \left(\{x\} - \tfrac{1}{2}\right) f'(x)\,dx &= (x - k - \tfrac{1}{2}) f(x)\Big|_k^{k+1} - \int_k^{k+1} f(x)\,dx \\ &= \tfrac{1}{2}\big(f(k+1) + f(k)\big) - \int_k^{k+1} f(x)\,dx. \end{aligned} \tag{2}$$

(Мы применили формулу интегрирования по частям.) Складывая обе части этого равенства для $1 \le k < n$, находим, что

$$\int_1^n \left(\{x\} - \tfrac{1}{2}\right) f'(x)\,dx = \sum_{1 \le k < n} f(k) + \tfrac{1}{2}\big(f(n) - f(1)\big) - \int_1^n f(x)\,dx,$$

т. е.

$$\sum_{1\le k<n} f(k) = \int_1^n f(x)\,dx - \tfrac{1}{2}\big(f(n)-f(1)\big) + \int_1^n B_1(\{x\})f'(x)\,dx, \tag{3}$$

где $B_1(x)$ — многочлен $B_1(x) = x - \frac{1}{2}$. Это и есть искомая формула, связывающая сумму с интегралом.

Продолжая интегрировать по частям, можно получать более точные приближения. Но прежде чем это сделать, рассмотрим *числа Бернулли*, которые являются коэффициентами следующего бесконечного ряда:

$$\frac{z}{e^z-1} = B_0 + B_1 z + \frac{B_2 z^2}{2!} + \cdots = \sum_{k\ge 0} \frac{B_k z^k}{k!}. \tag{4}$$

Коэффициенты этого ряда, которые встречаются во многих задачах, были введены Якобом Бернулли (Jacques Bernoulli) в работе *Ars Conjectandi*, опубликованной после его смерти в 1713 году. Самое интересное, что почти в то же самое время данные числа были открыты и японцем Такакузу Секи (Takakazu Seki) и впервые опубликованы в 1712 году, вскоре после *его* смерти. [См. *Takakazu Seki's Collected Works* (Osaka, 1974), 39–42.]

Имеем

$$B_0 = 1, \quad B_1 = -\tfrac{1}{2}, \quad B_2 = \tfrac{1}{6}, \quad B_3 = 0, \quad B_4 = -\tfrac{1}{30}; \tag{5}$$

некоторые последующие значения чисел Бернулли приведены в приложении А. Так как функция

$$\frac{z}{e^z-1} + \frac{z}{2} = \frac{z}{2}\,\frac{e^z+1}{e^z-1} = -\frac{z}{2}\,\frac{e^{-z}+1}{e^{-z}-1}$$

является четной, то

$$B_3 = B_5 = B_7 = B_9 = \cdots = 0. \tag{6}$$

Умножая обе части равенства (4) на $e^z - 1$ и приравнивая коэффициенты при одинаковых степенях z, получим формулу

$$\sum_k \binom{n}{k} B_k = B_n + \delta_{n1}. \tag{7}$$

(См. упр. 1.) Теперь определим *многочлен Бернулли*

$$B_m(x) = \sum_k \binom{m}{k} B_k x^{m-k}. \tag{8}$$

Если $m = 1$, то $B_1(x) = B_0 x + B_1 = x - \frac{1}{2}$; этот многочлен использовался выше, в соотношении (3). Если $m > 1$, то согласно (7) $B_m(1) = B_m = B_m(0)$; другими словами, $B_m(\{x\})$ не имеет разрывов при целых значениях x.

Вскоре станет ясно, какое отношение к нашей теме имеют многочлены Бернулли и числа Бернулли. Дифференцируя (8), находим

$$\begin{aligned} B_m'(x) &= \sum_k \binom{m}{k}(m-k)B_k x^{m-k-1} \\ &= m\sum_k \binom{m-1}{k} B_k x^{m-1-k} \\ &= mB_{m-1}(x) \end{aligned} \tag{9}$$

и, следовательно, при $m \geq 1$ можем проинтегрировать по частям:

$$\frac{1}{m!}\int_1^n B_m(\{x\})f^{(m)}(x)\,dx = \frac{1}{(m+1)!}\big(B_{m+1}(1)f^{(m)}(n) - B_{m+1}(0)f^{(m)}(1)\big) - \frac{1}{(m+1)!}\int_1^n B_{m+1}(\{x\})f^{(m+1)}(x)\,dx.$$

С помощью этого результата можно улучшить формулу (3) и, воспользовавшись (6), получить общую формулу Эйлера

$$\begin{aligned}\sum_{1\le k<n} f(k) &= \int_1^n f(x)\,dx - \frac{1}{2}\big(f(n)-f(1)\big) + \frac{B_2}{2!}\big(f'(n)-f'(1)\big) + \cdots \\ &\quad + \frac{(-1)^m B_m}{m!}\big(f^{(m-1)}(n) - f^{(m-1)}(1)\big) + R_{mn} \\ &= \int_1^n f(x)\,dx + \sum_{k=1}^{m}\frac{B_k}{k!}\big(f^{(k-1)}(n) - f^{(k-1)}(1)\big) + R_{mn}, \end{aligned} \tag{10}$$

где

$$R_{mn} = \frac{(-1)^{m+1}}{m!}\int_1^n B_m(\{x\})f^{(m)}(x)\,dx. \tag{11}$$

Остаток R_{mn} будет мал при очень малых значениях $B_m(\{x\})f^{(m)}(x)/m!$, и фактически можно показать, что для четного m

$$\left|\frac{B_m(\{x\})}{m!}\right| \le \frac{|B_m|}{m!} < \frac{4}{(2\pi)^m}. \tag{12}$$

[См. *CMath*, §9 5.] С другой стороны, обычно оказывается, что при увеличении m функция $f^{(m)}(x)$ возрастает по модулю, поэтому существует "наилучшее" значение m, при котором $|R_{mn}|$ принимает наименьшее значение (если n фиксировано)

Известно, что, когда m четно, существует число θ, такое, что

$$R_{mn} = \theta\frac{B_{m+2}}{(m+2)!}\big(f^{(m+1)}(n) - f^{(m+1)}(1)\big), \quad 0<\theta<1 \tag{13}$$

при условии, что $f^{(m+2)}(x)\,f^{(m+4)}(x) > 0$ для $1 < x < n$. В данном случае остаток меньше первого отбрасываемого члена и имеет такой же знак, как и у него Упрощенный вариант этого результата доказывается в упр. 3

А теперь применим формулу Эйлера к некоторым важным примерам Сначала положим $f(x) = 1/x$ Производные будут иметь вид $f^{(m)} = (-1)^m m!/x^{m+1}$, поэтому согласно (10) получим

$$H_{n-1} = \ln n + \sum_{k=1}^{m}\frac{B_k}{k}(-1)^{k-1}\left(\frac{1}{n^k} - 1\right) + R_{mn}. \tag{14}$$

Тогда

$$\gamma = \lim_{n\to\infty}(H_{n-1} - \ln n) = \sum_{k=1}^{m}\frac{B_k}{k}(-1)^k + \lim_{n\to\infty} R_{mn} \tag{15}$$

Из того, что существует предел $\lim_{n\to\infty} R_{mn} = \pm\int_1^\infty B_m(\{x\})\,dx/x^{m+1}$, следует, что существует константа γ. Тогда на основании (14) и (15) получаем общую приближенную формулу для гармонических чисел:

$$H_{n-1} = \ln n + \gamma + \sum_{k=1}^{m} \frac{(-1)^{k-1}B_k}{kn^k} + \int_n^\infty \frac{B_m(\{x\})\,dx}{x^{m+1}}$$

$$= \ln n + \gamma + \sum_{k=1}^{m-1} \frac{(-1)^{k-1}B_k}{kn^k} + O\Big(\frac{1}{n^m}\Big).$$

Заменив m на $m+1$, получим

$$H_{n-1} = \ln n + \gamma + \sum_{k=1}^{m} \frac{(-1)^{k-1}B_k}{kn^k} + O\Big(\frac{1}{n^{m+1}}\Big). \tag{16}$$

Более того, из (13) видно, что погрешность меньше первого отбрасываемого члена. В качестве частного случая (добавляя к обеим частям $1/n$) получим

$$H_n = \ln n + \gamma + \frac{1}{2n} - \frac{1}{12n^2} + \frac{1}{120n^4} - \epsilon, \quad 0 < \epsilon < \frac{B_6}{6n^6} = \frac{1}{252n^6}.$$

Это соотношение 1.2.7–(3). Для больших k числа Бернулли B_k становятся очень большими (приблизительно $(-1)^{1+k/2}2\big(k!/(2\pi)^k\big)$, если k четно), поэтому ни при каком фиксированном значении n ряд, полученный из (16), при $m\to\infty$ сходиться не будет.

Данный метод можно применить и для вывода приближенной формулы Стирлинга. На этот раз положим $f(x) = \ln x$ и из (10) получим

$$\ln(n-1)! = n\ln n - n + 1 - \tfrac{1}{2}\ln n + \sum_{1<k\le m} \frac{B_k(-1)^k}{k(k-1)}\Big(\frac{1}{n^{k-1}} - 1\Big) + R_{mn}. \tag{17}$$

Продолжая рассуждения, как было показано выше, приходим к выводу, что предел

$$\lim_{n\to\infty}(\ln n! - n\ln n + n - \tfrac{1}{2}\ln n) = 1 + \sum_{1<k\le m} \frac{B_k(-1)^{k+1}}{k(k-1)} + \lim_{n\to\infty} R_{mn}$$

существует. Временно обозначим его через σ ("постоянная Стирлинга"). В результате получим приближенную формулу Стирлинга

$$\ln n! = (n+\tfrac{1}{2})\ln n - n + \sigma + \sum_{1<k\le m} \frac{B_k(-1)^k}{k(k-1)n^{k-1}} + O\Big(\frac{1}{n^m}\Big). \tag{18}$$

В частности, положим $m = 5$. Тогда

$$\ln n! = (n+\tfrac{1}{2})\ln n - n + \sigma + \frac{1}{12n} - \frac{1}{360n^3} + O\Big(\frac{1}{n^5}\Big).$$

И теперь после потенцирования обеих частей находим:

$$n! = e^\sigma\sqrt{n}\Big(\frac{n}{e}\Big)^n \exp\Big(\frac{1}{12n} - \frac{1}{360n^3} + O\Big(\frac{1}{n^5}\Big)\Big).$$

Воспользовавшись тем, что $e^{\sigma} = \sqrt{2\pi}$ (см. упр. 5), и разложив экспоненту в ряд, получим окончательный результат:

$$n! = \sqrt{2\pi n}\left(\frac{n}{e}\right)^n\left(1 + \frac{1}{12n} + \frac{1}{288n^2} - \frac{139}{51840n^3} - \frac{571}{2488320n^4} + O\left(\frac{1}{n^5}\right)\right). \qquad \textbf{(19)}$$

УПРАЖНЕНИЯ

1. [*M18*] Докажите соотношение (7).

2. [*HM20*] Заметьте, что формула (9) следует из (8) для *любой* последовательности B_n, а не только для той, которая определяется соотношением (4). Объясните, почему использование последней является необходимым условием справедливости соотношения (10).

3. [*HM20*] Пусть $C_{mn} = ((-1)^m B_m/m!)(f^{(m-1)}(n) - f^{(m-1)}(1))$ — m-й корректирующий член в формуле суммирования Эйлера. Считая, что функция $f^{(m)}(x)$ имеет постоянный знак на промежутке $1 \le x \le n$, докажите, что $|R_{mn}| \le |C_{mn}|$ при $m = 2k > 0$; другими словами, покажите, что значение остатка по модулю не больше значения последнего вычисленного члена

▶ **4.** [*HM20*] (*Суммы степеней.*) Если $f(x) = x^m$, то все производные функции f порядка $m+1$ и выше равны нулю, поэтому формула суммирования Эйлера дает *точное* значение суммы

$$S_m(n) = \sum_{0 \le k < n} k^m,$$

выраженное с помощью чисел Бернулли (Именно изучая суммы $S_m(n)$ для $m = 1$, 2, 3, ..., Бернулли и Секи пришли к открытию этих чисел.) Представьте $S_m(n)$ с помощью *многочленов* Бернулли Проверьте полученный результат для $m = 0$, 1 и 2. (Обратите внимание, что суммирование в искомой сумме выполняется по $0 \le k < n$, а не по $1 \le k < n$; в формуле суммирования Эйлера единицу везде можно заменить нулем.)

5. [*HM30*] На основании формулы

$$n! = \kappa\sqrt{n}\left(\frac{n}{e}\right)^n\left(1 + O\left(\frac{1}{n}\right)\right)$$

и с помощью произведения Валлиса (упр. 1.2.5–18) покажите, что $\kappa = \sqrt{2\pi}$. [*Указание.* Рассмотрите $\binom{2n}{n}$ для больших значений n.]

▶ **6.** [*HM30*] Покажите, что приближенная формула Стирлинга справедлива также для нецелых значений n:

$$\Gamma(x+1) = \sqrt{2\pi x}\left(\frac{x}{e}\right)^x\left(1 + O\left(\frac{1}{x}\right)\right), \quad x \ge a > 0.$$

[*Указание.* В формуле суммирования Эйлера положите $f(x) = \ln(x + c)$ и воспользуйтесь определением $\Gamma(x)$, которое дано в разделе 1.2.5.]

▶ **7.** [*HM32*] Чему равно приближенное значение $1^1 2^2 3^3 \ldots n^n$?

8. [*M23*] Найдите асимптотическое представление для $\ln(an^2+bn)!$ с абсолютной погрешностью $O(n^{-2})$ Воспользуйтесь полученным результатом для нахождения асимптотического представления для $\binom{cn^2}{n}/c^n\binom{n^2}{n}$ с относительной погрешностью $O(n^{-2})$, где c — положительная константа В данном случае под *абсолютной погрешностью* понимаем такое ϵ, которое удовлетворяет соотношению (точное значение) = (приближенное значение) $+ \epsilon$, а под *относительной погрешностью* — ϵ, удовлетворяющее соотношению (точное значение) = (приближенное значение)$(1 + \epsilon)$.

▸ **9.** [*M25*] Найдите асимптотическое представление для $\binom{2n}{n}$ с относительной погрешностью порядка $O(n^{-3})$ двумя способами. (a) с помощью приближенной формулы Стирлинга; (b) с помощью упр. 1.2.6–47 и формулы 1.2.11.1–(16).

***1.2.11.3. Применение асимптотических формул.** В данном разделе мы рассмотрим три следующие интересные суммы, чтобы найти их приближенные значения:

$$P(n) = 1 + \frac{n-1}{n} + \frac{n-2}{n}\frac{n-2}{n-1} + \cdots = \sum_{k=0}^{n} \frac{(n-k)^k (n-k)!}{n!}, \tag{1}$$

$$Q(n) = 1 + \frac{n-1}{n} + \frac{n-1}{n}\frac{n-2}{n} + \cdots = \sum_{k=1}^{n} \frac{n!}{(n-k)!\, n^k}, \tag{2}$$

$$R(n) = 1 + \frac{n}{n+1} + \frac{n}{n+1}\frac{n}{n+2} + \cdots = \sum_{k\geq 0} \frac{n!\, n^k}{(n+k)!}. \tag{3}$$

Эти функции, на первый взгляд, похожи, но на самом деле в корне отличаются одна от другой. Они возникают в некоторых алгоритмах, которые будут рассмотрены несколько позже. И $P(n)$, и $Q(n)$ — конечные суммы, в то время как $R(n)$ — бесконечная сумма. Может показаться, что при больших n значения всех трех сумм будут практически одинаковы, хотя совсем не очевидно, каким будет приближенное значение *каждой* из них в отдельности. В процессе поиска приближенных значений этих функций мы получим ряд очень поучительных побочных результатов. (Если хотите, можете временно прекратить чтение и попытаться самостоятельно исследовать эти функции, прежде чем вернуться к книге и выяснить, как с ними поступим мы.)

Прежде всего отметим важную связь между $Q(n)$ и $R(n)$:

$$\begin{aligned} Q(n) + R(n) &= \frac{n!}{n^n}\left(\left(1 + n + \cdots + \frac{n^{n-1}}{(n-1)!}\right) + \left(\frac{n^n}{n!} + \frac{n^{n+1}}{(n+1)!} + \cdots\right)\right) \\ &= \frac{n!\, e^n}{n^n}. \end{aligned} \tag{4}$$

Формула Стирлинга говорит о том, что $n!\, e^n/n^n$ приближенно равно $\sqrt{2\pi n}$, поэтому можно догадаться, что каждая из функций $Q(n)$ и $R(n)$ приближенно равна $\sqrt{\pi n/2}$.

Теперь, чтобы продвинуться дальше, рассмотрим частные суммы ряда для e^n. Воспользовавшись формулой Тейлора с остаточным членом

$$f(x) = f(0) + f'(0)x + \cdots + \frac{f^{(n)}(0)x^n}{n!} + \int_0^x \frac{t^n}{n!} f^{(n+1)}(x-t)\, dt, \tag{5}$$

мы вскоре поймем необходимость введения важной функции, которая называется *неполной гамма-функцией:*

$$\gamma(a, x) = \int_0^x e^{-t} t^{a-1}\, dt. \tag{6}$$

Будем считать, что $a > 0$. Из упр. 1.2.5–20 имеем $\gamma(a, \infty) = \Gamma(a)$; этим и объясняется название "неполная гамма-функция". Для нее существует два полезных разложения

в ряд по степеням x (см. упр. 2 и 3):

$$\gamma(a,x) = \frac{x^a}{a} - \frac{x^{a+1}}{a+1} + \frac{x^{a+2}}{2!\,(a+2)} - \cdots = \sum_{k\ge 0} \frac{(-1)^k x^{k+a}}{k!\,(k+a)}, \tag{7}$$

$$e^x\gamma(a,x) = \frac{x^a}{a} + \frac{x^{a+1}}{a(a+1)} + \frac{x^{a+2}}{a(a+1)(a+2)} + \cdots = \sum_{k\ge 0} \frac{x^{k+a}}{a(a+1)\ldots(a+k)}. \tag{8}$$

Из второй формулы видна связь с $R(n)$:

$$R(n) = \frac{n!\,e^n}{n^n}\left(\frac{\gamma(n,n)}{(n-1)!}\right). \tag{9}$$

Данное равенство специально было записано в более сложном виде, чем это необходимо, так как $\gamma(n,n)$ — часть $\gamma(n,\infty) = \Gamma(n) = (n-1)!$, а $n!\,e^n/n^n$ — величина из (4).

Поэтому задача сводится к нахождению хороших оценок для $\gamma(n,n)/(n-1)!$. Теперь определим приближенное значение $\gamma(x+1,\ x+y)/\Gamma(x+1)$ для фиксированного y и больших x. Заметим, что используемые здесь методы важнее результатов, поэтому читателю следует внимательно разобраться в приведенных ниже выкладках.

По определению имеем

$$\begin{aligned}\frac{\gamma(x+1,\,x+y)}{\Gamma(x+1)} &= \frac{1}{\Gamma(x+1)}\int_0^{x+y} e^{-t}t^x\,dt \\ &= 1 - \frac{1}{\Gamma(x+1)}\int_x^{\infty} e^{-t}t^x\,dt + \frac{1}{\Gamma(x+1)}\int_x^{x+y} e^{-t}t^x\,dt.\end{aligned} \tag{10}$$

Обозначим

$$I_1 = \int_x^{\infty} e^{-t}t^x\,dt,$$

$$I_2 = \int_x^{x+y} e^{-t}t^x\,dt$$

и рассмотрим по очереди каждый из этих интегралов.

Оценка I_1. Выполнив подстановку $t = x(1+u)$ получим интеграл от 0 до бесконечности. В результате дальнейшей подстановки $v = u - \ln(1+u)$, $dv = \bigl(1 - 1/(1+u)\bigr)\,du$ (она законна, так как v — монотонная функция от u) получаем:

$$I_1 = e^{-x}x^x\int_0^{\infty} xe^{-xu}(1+u)^x\,du = e^{-x}x^x\int_0^{\infty} xe^{-xv}\left(1+\frac{1}{u}\right)dv. \tag{11}$$

В последнем интеграле заменим $1 + 1/u$ рядом по степеням v. Тогда

$$v = \tfrac{1}{2}u^2 - \tfrac{1}{3}u^3 + \tfrac{1}{4}u^4 - \tfrac{1}{5}u^5 + \cdots = (u^2/2)(1 - \tfrac{2}{3}u + \tfrac{1}{2}u^2 - \tfrac{2}{5}u^3 + \cdots$$

Полагая $w = \sqrt{2v}$, получим

$$w = u(1 - \tfrac{2}{3}u + \tfrac{1}{2}u^2 - \tfrac{2}{5}u^3 + \cdots)^{1/2} = u - \tfrac{1}{3}u^2 + \tfrac{7}{36}u^3 - \tfrac{73}{540}u^4 + \tfrac{1331}{12960}u^5 + O(u^6).$$

(Это разложение можно получить на основании биномиальной теоремы. Эффективные методы выполнения подобных преобразований, а также других операций над

степенными рядами, которые понадобятся нам несколько позже, рассматриваются в разделе 4.7.) Теперь можно получить разложение u в ряд по степеням w:

$$u = w + \frac{1}{3}w^2 + \frac{1}{36}w^3 - \frac{1}{270}w^4 + \frac{1}{4320}w^5 + O(w^6);$$

$$1 + \frac{1}{u} = 1 + \frac{1}{w} - \frac{1}{3} + \frac{1}{12}w - \frac{2}{135}w^2 + \frac{1}{864}w^3 + O(w^4)$$

$$= \frac{1}{\sqrt{2}}v^{-1/2} + \frac{2}{3} + \frac{\sqrt{2}}{12}v^{1/2} - \frac{4}{135}v + \frac{\sqrt{2}}{432}v^{3/2} + O(v^2). \tag{12}$$

Во всех этих формулах O-запись относится к малым значениям аргумента, т. е. $|u| \le r$, $|v| \le r$, $|w| \le r$ для достаточно малого положительного r. Не слишком ли сильное это ограничение? Предполагается, что подстановка $1 + 1/u$ как функции от v в (11) законна и для $0 \le v < \infty$, а не только для $|v| \le r$. К счастью, оказывается, что значение интеграла от 0 до ∞ почти полностью зависит от значений подынтегральной функции в окрестности нуля. В самом деле, получаем (см. упр. 4)

$$\int_r^\infty xe^{-xv}\Big(1 + \frac{1}{u}\Big)dv = O(e^{-rx}) \tag{13}$$

для любого фиксированного $r > 0$ и для больших x. Нас интересует аппроксимация с точностью до членов порядка $O(x^{-m})$, и так как $O\big((1/e^r)^x\big)$ намного меньше, чем $O(x^{-m})$ для любых положительных r и m, нужно взять интеграл только от 0 до r для любого фиксированного положительного r. Поэтому возьмем достаточно малое r, чтобы все выполненные выше операции над степенными рядами были законны (см. соотношения 1.2.11.1–(11) и 1.2.11.3–(13)).

Так как

$$\int_0^\infty xe^{-xv}v^\alpha\,dv = \frac{1}{x^\alpha}\int_0^\infty e^{-q}q^\alpha\,dq = \frac{1}{x^\alpha}\Gamma(\alpha+1), \quad \text{если } \alpha > -1, \tag{14}$$

то, подставляя ряд (12) в интеграл (11), окончательно получаем

$$I_1 = e^{-x}x^x\Big(\sqrt{\frac{\pi}{2}}x^{1/2} + \frac{2}{3} + \frac{\sqrt{2\pi}}{24}x^{-1/2} - \frac{4}{135}x^{-1} + \frac{\sqrt{2\pi}}{576}x^{-3/2} + O(x^{-2})\Big). \tag{15}$$

Оценка I_2. В интеграле I_2 делаем подстановку $t = u + x$ и получаем

$$I_2 = e^{-x}x^x\int_0^y e^{-u}\Big(1 + \frac{u}{x}\Big)^x\,du. \tag{16}$$

Теперь

$$e^{-u}\Big(1 + \frac{u}{x}\Big)^x = \exp\Big(-u + x\ln\Big(1 + \frac{u}{x}\Big)\Big) = \exp\Big(\frac{-u^2}{2x} + \frac{u^3}{3x^2} + O(x^{-3})\Big)$$

$$= 1 - \frac{u^2}{2x} + \frac{u^4}{8x^2} + \frac{u^3}{3x^2} + O(x^{-3})$$

для $0 \le u \le y$ и больших x. Поэтому

$$I_2 = e^{-x}x^x\left(y - \frac{y^3}{6}x^{-1} + \Big(\frac{y^4}{12} + \frac{y^5}{40}\Big)x^{-2} + O(x^{-3})\right). \tag{17}$$

И в заключение исследуем множитель $e^{-x}x^x/\Gamma(x+1)$, который появляется при умножении (15) и (17) на коэффициент $1/\Gamma(x+1)$ из (10). По формуле Стирлинга, которая согласно упр. 1.2.11.2–6 справедлива для гамма-функции, имеем

$$\frac{e^{-x}x^x}{\Gamma(x+1)} = \frac{e^{-1/12x+O(x^{-3})}}{\sqrt{2\pi x}}$$
$$= \frac{1}{\sqrt{2\pi}}x^{-1/2} - \frac{1}{12\sqrt{2\pi}}x^{-3/2} + \frac{1}{288\sqrt{2\pi}}x^{-5/2} + O(x^{-7/2}). \qquad (18)$$

А теперь из соотношений (10), (15), (17) и (18) получаем следующую теорему.

Теорема А. *Для больших значений x и фиксированного y*

$$\frac{\gamma(x+1.\,x+y)}{\Gamma(x+1)} = \frac{1}{2} + \left(\frac{y-2/3}{\sqrt{2\pi}}\right)x^{-1/2} + \frac{1}{\sqrt{2\pi}}\left(\frac{23}{270} - \frac{y}{12} - \frac{y^3}{6}\right)x^{-3/2}$$
$$+ O(x^{-5/2}). \quad \blacksquare \qquad (19)$$

Примененный метод показывает, что данное разложение можно продолжить настолько, насколько это необходимо, и получить приближенную формулу для последующих степеней x.

Теорему А можно использовать для получения приближенных значений $R(n)$ и $Q(n)$ с помощью (4) и (9), но мы сделаем это несколько позже. А пока давайте вернемся к сумме $P(n)$, для изучения которой, похоже, необходим немного другой метод. Имеем

$$P(n) = \sum_{k=0}^{n} \frac{k^{n-k}k!}{n!} = \frac{\sqrt{2\pi}}{n!}\sum_{k=0}^{n} k^{n+1/2}e^{-k}\left(1 + \frac{1}{12k} + O(k^{-2})\right). \qquad (20)$$

Таким образом, для получения значений $P(n)$ требуется изучить суммы вида

$$\sum^{n} k^{n+1/2}e^{-k}.$$

Положим $f(x) = x^{n+1/2}e^{-x}$ и применим формулу суммирования Эйлера:

$$\sum_{k=0}^{n} k^{n+1/2}e^{-k} = \int_0^n x^{n+1/2}e^{-x}\,dx + \tfrac{1}{2}n^{n+1/2}e^{-n} + \tfrac{1}{24}n^{n-1/2}e^{-n} - R. \qquad (21)$$

Предварительный анализ остаточного члена (см. упр. 5) показывает, что $R = O(n^ne^{-n})$; и так как интеграл является неполной гамма-функцией, имеем

$$\sum_{k=0}^{n} k^{n+1/2}e^{-k} = \gamma\left(n+\tfrac{3}{2},\, n\right) + \tfrac{1}{2}n^{n+1/2}e^{-n} + O(n^ne^{-n}). \qquad (22)$$

Для формулы (20) требуется еще найти оценку суммы

$$\sum_{k=0}^{n} k^{n-1/2}e^{-k} = \sum_{0\le k\le n-1} k^{(n-1)+1/2}e^{-k} + n^{n-1/2}e^{-n},$$

и это также можно получить с помощью соотношения (22).

Теперь в нашем распоряжении достаточно формул для того, чтобы определить приближенные значения $P(n)$, $Q(n)$ и $R(n)$; осталось только подставить, умножить и т. д. При этом нам представится случай воспользоваться разложением

$$(n+\alpha)^{n+\beta} = n^{n+\beta}e^{\alpha}\left(1+\alpha\left(\beta-\frac{\alpha}{2}\right)\frac{1}{n}+O(n^{-2})\right), \tag{23}$$

которое доказывается в упр. 6. Метод, который мы использовали в (21), дает только первые два члена асимптотического ряда для $P(n)$. Следующие члены можно получить с помощью важного метода, описанного в упр. 14.

В результате всех этих рассуждений получаем нужные асимптотические формулы:

$$P(n) = \sqrt{\frac{\pi n}{2}} - \frac{2}{3} + \frac{11}{24}\sqrt{\frac{\pi}{2n}} + \frac{4}{135n} - \frac{71}{1152}\sqrt{\frac{\pi}{2n^3}} + O(n^{-2}), \tag{24}$$

$$Q(n) = \sqrt{\frac{\pi n}{2}} - \frac{1}{3} + \frac{1}{12}\sqrt{\frac{\pi}{2n}} - \frac{4}{135n} + \frac{1}{288}\sqrt{\frac{\pi}{2n^3}} + O(n^{-2}), \tag{25}$$

$$R(n) = \sqrt{\frac{\pi n}{2}} + \frac{1}{3} + \frac{1}{12}\sqrt{\frac{\pi}{2n}} + \frac{4}{135n} + \frac{1}{288}\sqrt{\frac{\pi}{2n^3}} + O(n^{-2}). \tag{26}$$

Изученные нами функции только поверхностно рассматривались в опубликованных научных работах. Первый член $\sqrt{\pi n/2}$ в разложении суммы $P(n)$ был получен Г. Б. Демутом (H. B. Demuth) [Ph. D. thesis (Stanford University, October, 1956), 67–68]. Используя данный результат, таблицу значений $P(n)$ для $n \le 2000$ и хорошую логарифмическую линейку, автор в 1963 году продолжил эти исследования и получил эмпирическую оценку $P(n) \approx \sqrt{\pi n/2} - 0.6667 + 0.575/\sqrt{n}$. Было естественно предположить, что на втором месте стоит $2/3$, 0.6667 является приближенным значением для $2/3$, а 0.575, вероятно, будет приближением для $\gamma = 0.57721\ldots$, которое должно заменять это число в третьем слагаемом (почему бы не быть оптимистом?). Впоследствии, во время написания данного раздела, было найдено верное разложение для $P(n)$ и предположение о том, что 0.6667 нужно заменить на $2/3$, подтвердилось. Что же касается коэффициента 0.575, то он является приближением не для γ, а для $\frac{11}{24}\sqrt{\pi/2} \approx 0.5744$. Это отлично подтверждает и теорию, и эмпирические оценки.

Эквиваленты асимптотических формул для $Q(n)$ и $R(n)$ были впервые определены выдающимся индийским математиком-самоучкой С. Рамануджаном (S. Ramanujan), который поставил задачу оценки $n!\,e^n/2n^n - Q(n)$ в *J. Indian Math. Soc.* **3** (1911), 128; **4** (1912), 151–152. В качестве решения этой задачи он предложил асимптотический ряд

$$\frac{1}{3} + \frac{4}{135}n^{-1} - \frac{8}{2835}n^{-2} - \frac{1}{8505}n^{-3} + \cdots,$$

который гораздо лучше соотношения (25). По сравнению с приведенным выше методом доказательство Рамануджана было более изящным. Оценивая I_1, он выполнил подстановку $t = x + u\sqrt{2x}$ и выразил подынтегральное выражение в виде суммы членов вида $c_{jk}\int_0^\infty \exp(-u^2)u^j x^{-k/2}\,du$. Интеграл I_2 можно вообще не рассматривать,

так как $a\gamma(a,x) = x^a e^{-x} + \gamma(a+1,x)$, где $a > 0$ (см. (8)). Еще более простой — вероятно, самый простой из возможных — метод получения асимптотической формулы для $Q(n)$ приведен в упр. 20. Несмотря на излишнюю сложность, использованный нами метод все же является очень поучительным. Он был предложен Р Фюрхом (R. Furch) [*Zeitschrift für Physik* **112** (1939), 92–95], который главным образом, интересовался значением y. удовлетворяющим соотношению $\gamma(x+1, x+y) = \Gamma(x+1)/2$ Асимптотические свойства неполной гамма-функции были впоследствии обобщены для комплексного аргумента Ф. Дж. Трикоми (F. G. Tricomi) [*Math. Zeitschrift* **53** (1950), 136–148]. См. также N. M. Temme, *Math. Comp* **29** (1975), 1109–1114; *SIAM J. Math. Anal.* **10** (1979), 757–766 Ссылки на некоторые другие исследования суммы $Q(n)$ перечислены в работе H. W. Gould, *AMM* **75** (1968), 1019–1021.

При выводе асимптотических формул для сумм $P(n)$, $Q(n)$ и $R(n)$ мы использовали только элементарные преобразования; но обратите внимание, что для каждой функции мы использовали свой метод! На самом деле во всех трех случаях можно было бы воспользоваться методом из упр. 14, который обсуждается в разделах 5.1.4 и 5.2.2. Это было бы более изящно, но менее поучительно.

Дополнительную информацию по этой теме можно найти в прекрасной книге Н. Г. де Брейна (N. G. de Bruijn) *Asymptotic Methods in Analysis* (Amsterdam: North-Holland, 1961) (Н. Г. де Брейн, *Асимптотические методы в анализе.* — М.: Изд-во иностр. лит., 1961). За более свежей информацией обратитесь к работе A. M. Odlyzko, *Handbook of Combinatorics* **2** (MIT Press, 1995), 1063–1229, в которой содержится 65 подробных примеров и подробная библиография.

УПРАЖНЕНИЯ

1. [*HM20*] Докажите формулу (5) индукцией по n.

2. [*HM20*] Выведите (7) из (6).

3. [*M20*] Выведите (8) из (7).

▶ **4.** [*HM10*] Докажите соотношение (13).

5. [*HM24*] Покажите, что в соотношении (21) R имеет порядок $O(n^n e^{-n})$.

▶ **6.** [*HM20*] Докажите соотношение (23).

▶ **7.** [*HM30*] Оценивая I_2, мы рассматривали интеграл $\int_0^y e^{-u}\left(1+\frac{u}{x}\right)^x du$. Дайте асимптотическое представление

$$\int_0^{yx^{1/4}} e^{-u}\left(1+\frac{u}{x}\right)^x du$$

с точностью до членов порядка $O(x^{-2})$ для фиксированного y и больших x

8. [*HM30*] Пусть $f(x) = O(x^r)$ при $x \to \infty$ и $0 \le r < 1$ Покажите, что

$$\int_0^{f(x)} e^{-u}\left(1+\frac{u}{x}\right)^x du = \int_0^{f(x)} \exp\left(\frac{-u^2}{2x} + \frac{u^3}{3x^2} - \cdots + \frac{(-1)^{m-1}u^m}{mx^{m-1}}\right) du + O(x^{-s}).$$

если $m = \lceil (s+2r)/(1-r) \rceil$. [Отсюда, как частный случай, следует результат, принадлежащий Трикоми (Tricomi) если $f(x) = O(\sqrt{x})$, то

$$\int_0^{f(x)} e^{-u}\left(1+\frac{u}{x}\right)^x du = \sqrt{2x}\int_0^{f(x)/\sqrt{2x}} e^{-t^2} dt + O(1)\,]$$

▸ **9.** [*HM36*] Как ведет себя функция $\gamma(x+1, px)/\Gamma(x+1)$ при больших x? (Здесь p — действительная константа; и если $p<0$, мы считаем, что x — целое, чтобы t^x было определено и для отрицательных t.) Найдите по меньшей мере два члена асимптотического представления, прежде чем использовать символ O.

10. [*HM34*] При тех же предположениях, что и в предыдущем упражнении, при $p \neq 1$, найдите для фиксированного y асимптотическое представление функции

$$\gamma(x+1, px+py/(p-1)) - \gamma(x+1, px)$$

с точностью до членов того же порядка, что и в предыдущем упражнении.

▸ **11.** [*HM35*] Обобщим функции $Q(n)$ и $R(n)$, введя параметр x:

$$Q_x(n) = 1 + \frac{n-1}{n}x + \frac{n-1}{n}\frac{n-2}{n}x^2 + \cdots,$$
$$R_x(n) = 1 + \frac{n}{n+1}x + \frac{n}{n+1}\frac{n}{n+2}x^2 + \cdots.$$

Исследуйте эти функции и найдите для них асимптотические формулы при $x \neq 1$.

12. [*HM20*] Функцию $\int_0^x e^{-t^2/2}\,dt$, которая появляется в связи с нормальным распределением (см. раздел 1.2.10), можно представить в виде частного случая неполной гамма-функции. Найдите значения a, b и y, такие, что $b\gamma(a,y)$ равно $\int_0^x e^{-t^2/2}\,dt$.

13. [*HM42*] (С. Рамануджан.) Докажите, что $R(n) - Q(n) = \frac{2}{3} + 8/(135(n+\theta(n)))$, где $\frac{2}{21} \le \theta(n) \le \frac{8}{45}$. (Отсюда следует более слабый результат: $R(n+1)-Q(n+1) < R(n)-Q(n)$.)

▸ **14.** [*HM39*] (Н. Г. де Брейн.) Цель данного упражнения — найти асимптотическое представление суммы $\sum_{k=0}^n k^{n+\alpha}e^{-k}$ для фиксированного α при $n \to \infty$.

a) Заменив k на $n-k$, покажите, что данная сумма равна $n^{n+\alpha}e^{-n}\sum_{k=0}^n e^{-k^2/2n}f(k,n)$, где

$$f(k,n) = \Big(1-\frac{k}{n}\Big)^{\alpha}\exp\Big(-\frac{k^3}{3n^2} - \frac{k^4}{4n^3} - \cdots\Big).$$

b) Покажите, что для всех $m \ge 0$ и $\epsilon > 0$ величину $f(k,n)$ можно записать в виде

$$\sum_{0\le i\le j\le m} c_{ij}k^{2i+j}n^{-i-j} + O(n^{(m+1)(-1/2+3\epsilon)}), \quad \text{если } 0 \le k \le n^{1/2+\epsilon}.$$

c) Докажите, что как следствие из (b) имеем

$$\sum_{k=0}^n e^{-k^2/2n}f(k,n) = \sum_{0\le i\le j\le m} c_{ij}n^{-i-j}\sum_{k\ge 0} k^{2i+j}e^{-k^2/2n} + O(n^{-m/2+\delta})$$

для всех $\delta > 0$. [*Указание.* Суммы по k, где $n^{1/2+\epsilon} < k < \infty$, равны $O(n^{-r})$ для всех r.]

d) Покажите, что асимптотическое представление суммы $\sum_{k\ge 0} k^t e^{-k^2/2n}$ для фиксированного $t \ge 0$ можно получить с помощью формулы суммирования Эйлера.

e) И наконец, покажите, что

$$\sum_{k=0}^n k^{n+\alpha}e^{-k} = n^{n+\alpha}e^{-n}\left(\sqrt{\frac{\pi n}{2}} - \frac{1}{6} - \alpha + \left(\frac{1}{12} + \frac{1}{2}\alpha + \frac{1}{2}\alpha^2\right)\sqrt{\frac{\pi}{2n}} + O(n^{-1})\right)$$

эту формулу можно получить с точностью до членов порядка $O(n^{-r})$ для любого r.

15. [*HM20*] Найдите связь между интегралом

$$\int_0^\infty \Big(1+\frac{z}{n}\Big)^n e^{-z}\,dz$$

и функцией $Q(n)$.

16. [*M24*] Докажите тождество

$$\sum_k (-1)^k \binom{n}{k} k^{n-1} Q(k) = (-1)^n (n-1)!, \quad \text{где } n > 0.$$

17. [*HM29*] (К. В. Миллер (K. W. Miller).) Из соображений симметрии рассмотрим также четвертый ряд, который является для $P(n)$ тем, чем $R(n)$ является для $Q(n)$:

$$S(n) = 1 + \frac{n}{n+1} + \frac{n}{n+2}\,\frac{n+1}{n+2} + \cdots = \sum_{k\ge 0} \frac{(n+k-1)!}{(n-1)!\,(n+k)^k}.$$

Как выглядит асимптотическое представление этой функции?

18. [*M25*] Покажите, что суммы $\sum \binom{n}{k} k^k (n-k)^{n-k}$ и $\sum \binom{n}{k} (k+1)^k (n-k)^{n-k}$ можно очень просто выразить через функцию Q.

19. [*HM30*] (*Лемма Ватсона (Watson).*) Покажите, что если интеграл

$$C_n = \int_0^\infty e^{-nx} f(x)\,dx$$

существует для всех больших n и если $f(x) = O(x^\alpha)$ для $0 \le x \le r$, где $r > 0$ и $\alpha > -1$, то $C_n = O(n^{-1-\alpha})$.

▶ **20.** [*HM30*] Пусть $u = w + \frac{1}{3}w^2 + \frac{1}{36}w^3 - \frac{1}{270}w^4 + \cdots = \sum_{k=1}^\infty c_k w^k$ — степенной ряд, который является решением уравнения $w = (u^2 - \frac{2}{3}u^3 + \frac{2}{4}u^4 - \frac{2}{5}u^5 + \cdots)^{1/2}$ (см. (12)). Покажите, что

$$Q(n) + 1 = \sum_{k=1}^{m-1} k c_k \Gamma(k/2) \left(\frac{n}{2}\right)^{1-k/2} + O(n^{1-m/2})$$

для всех $m \ge 1$. [*Указание.* Примените лемму Ватсона к тождеству из упр. 15.]

Я чувствую, что должна добиться успеха в математике,
хотя и не понимаю, почему это так важно.

— ХЕЛЕН КЕЛЛЕР (HELEN KELLER) (1898)

1.3. MIX

В этой книге ОЧЕНЬ ЧАСТО встречаются упоминания о внутреннем машинном языке компьютера. Причем использовать мы будем гипотетический компьютер под названием "MIX". MIX практически ничем не отличается от любого другого компьютера 60–70-х годов, только он, вероятно, более изящен. При разработке языка компьютера MIX преследовалась цель сделать его достаточно мощным, позволяющим для большинства алгоритмов писать короткие программы, и в то же время достаточно простым, чтобы его операции можно было легко запомнить

Настоятельно рекомендую читателю внимательно изучить этот раздел, так как язык MIX используется в очень многих разделах книги. Отбросьте все сомнения по поводу того, стоит ли изучать машинный язык. Автор однажды понял, что нет ничего необычного в том, чтобы в течение одной недели заниматься написанием программ на нескольких различных машинных языках! Каждый, кто серьезно интересуется компьютерами, должен рано или поздно изучить по крайней мере один машинный язык. При разработке MIX были специально сохранены основные черты реальных компьютеров, чтобы его характеристики можно было легко понять и усвоить.

Тем не менее нужно признать, что в настоящее время MIX полностью устарел. Поэтому в последующих изданиях данной книги он будет заменен новым компьютером под названием MMIX (номер 2009). MMIX будет представлять собой так называемый компьютер с усеченным набором команд (RISC), который выполняет арифметические операции над 64-битовыми словами. Будучи более изящным, чем MIX, он станет аналогом тех компьютеров, которые в 90-х годах завоевали ключевые позиции на рынке компьютерной техники.

Полный и повсеместный переход в данной книге от MIX к MMIX отнимет много времени, поэтому огромная просьба к добровольцам — окажите посильную помощь в этом деле. Между тем, автор надеется, что читатели согласятся подождать еще несколько лет и пока удовлетворятся устаревшей архитектурой MIX, которая все еще заслуживает внимания, поскольку обеспечивает среду для дальнейших разработок.

1.3.1. Описание MIX

MIX — это первый в мире полиненасыщенный компьютер*. Как и у большинства компьютеров, у него есть идентификационный номер — 1009. Этот номер получен следующим образом: взяли 16 очень похожих на MIX реальных компьютеров, на которых можно легко имитировать MIX, а затем нашли среднее значение их номеров, взятых с равными весовыми коэффициентами:

$$\lfloor(360 + 650 + 709 + 7070 + \mathrm{U3} + \mathrm{SS80} + 1107 + 1604 + \mathrm{G20} + \mathrm{B220} + \mathrm{S2000} + 920 + 601 + \mathrm{H800} + \mathrm{PDP\text{-}4} + \mathrm{II})/16\rfloor = 1009. \quad (1)$$

Это же число можно получить значительно проще — прочитать слово MIX как римское число.

Характерная особенность компьютера MIX состоит в том, что он является двоичным и десятичным одновременно. *Программисты MIX на самом деле даже не знают,*

* Аналогия с полиненасыщенными жирами, широко рекламируемыми сегодня по всему миру. — *Прим. перев.*

компьютер с какой арифметикой они программируют — с двоичной или десятичной Поэтому алгоритмы, написанные для MIX, с небольшими изменениями можно использовать на любом из этих типов компьютеров и MIX можно легко имитировать на этих компьютерах. Те программисты, которые привыкли к двоичному компьютеру, могут считать MIX двоичным, а те, которые привыкли к десятичному, могут считать MIX десятичным. Программисты же с другой планеты могут считать MIX троичным компьютером.

Слова. Основной единицей информации является *байт*. Каждый байт должен принимать по меньшей мере 64 различных значения, но реальный объем содержащейся в байте информации может быть *разным*. Таким образом, в одном байте может содержаться любое число от 0 до 63 включительно. Более того, в каждом байте может содержаться *максимум* 100 различных значений. Следовательно, в двоичном компьютере байт должен состоять из шести разрядов, а в десятичном — из двух*.

Программы на языке MIX должны быть написаны так, чтобы в байте содержалось *не более* 64 значений. Так, для представления числа 80 мы всегда будем выделять два байта, хотя в десятичном компьютере для этого достаточно одного байта. *Алгоритм на языке MIX должен работать правильно независимо от размера байта.* Конечно, вполне возможно написать программы, зависящие от размера байта, но в данной книге такие действия осуждаются и допустимыми считаются только те программы, которые дают правильный результат независимо от размера байта Обычно придерживаться этого основного правила совсем нетрудно, и, таким образом, мы обнаружим, что программирование на десятичном компьютере не особенно отличается от программирования на двоичном.

С помощью двух соседних байтов можно выразить числа от 0 до 4 095.

С помощью трех соседних байтов можно выразить числа от 0 до 262 143.

С помощью четырех соседних байтов можно выразить числа от 0 до 16 777 215.

С помощью пяти соседних байтов можно выразить числа от 0 до 1 073 741 823.

Машинное слово состоит из пяти байтов и знака. Знак может принимать только два значения: "+" и "−".

Регистры. В компьютере MIX всего девять регистров (рис. 13).

Регистр A (аккумулятор) содержит 5 байт и знак.

Регистр X (расширение аккумулятора) тоже содержит 5 байт и знак

В регистрах I (индексных регистрах) I1, I2, I3, I4, I5 и I6 содержится по два байта и знак.

Регистр J (адрес перехода) содержит два байта; его знак — всегда "+".

Для обозначения регистра компьютера MIX будем использовать в качестве приставки к имени регистра строчную букву "r". Таким образом, "rA" обозначает "регистр A".

Регистр A имеет много применений, особенно часто он используется при выполнении арифметических действий и операций над данными. Регистр X используется

* Приблизительно с 1975 года слово "байт" стало обозначать последовательность, состоящую ровно из восьми двоичных цифр, что позволяет представлять числа от 0 до 255 Поэтому размеры байтов реальных компьютеров больше, чем размеры байтов гипотетической машины MIX И в самом деле, старомодные байты компьютера MIX только чуть-чуть больше, чем половина байта реального компьютера Говоря о байтах применительно к MIX, мы будем придерживаться прежнего значения этого слова, вновь возвращаясь к тем дням, когда понятие байта еще не было так стандартизовано

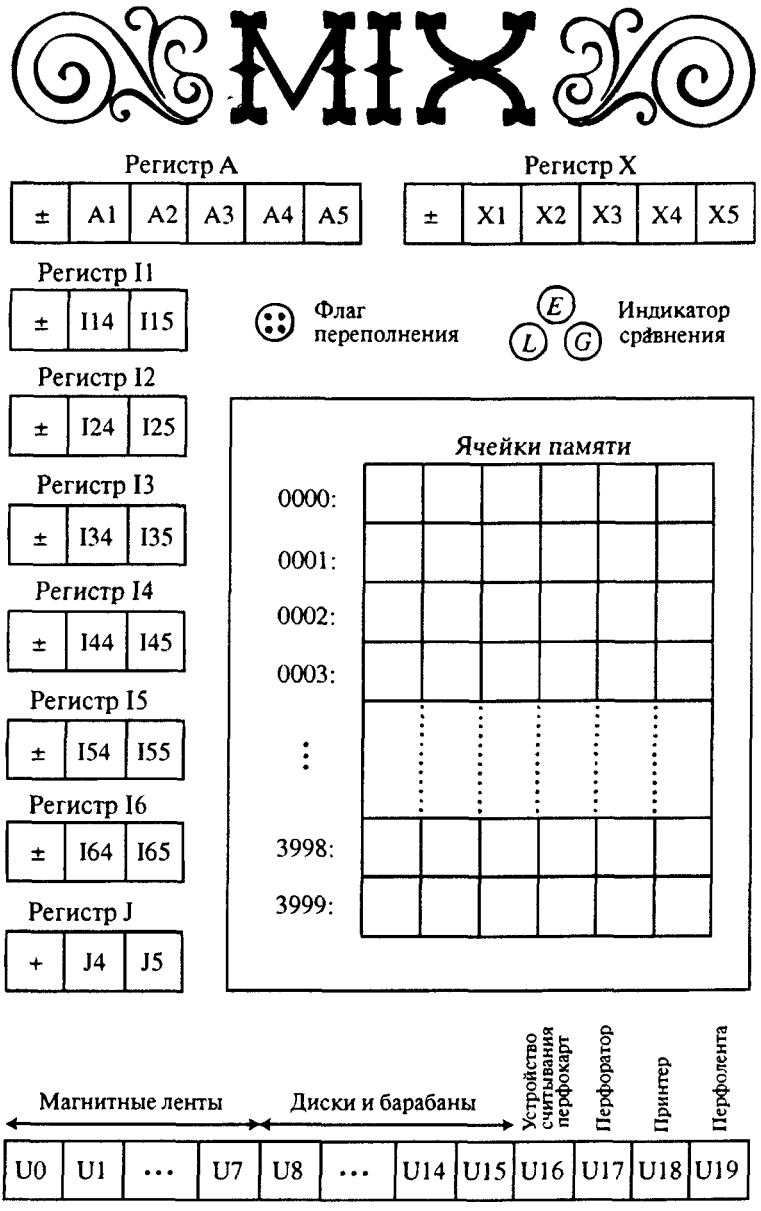


Рис. 13. Компьютер `MIX`.

для расширения вправо регистра A и вместе с rA — для хранения 10 байт произведения или делимого. Он может применяться и для хранения информации, сдвинутой вправо из rA. Индексные регистры rI1, rI2, rI3, rI4, rI5 и rI6 используются, главным образом, как счетчики и для ссылок на различные адреса памяти. В регистре J всегда хранится адрес команды, которая следует за последней операцией "перехода"; этот регистр используется, главным образом, для вызова подпрограмм.

Помимо регистров, компьютер MIX содержит следующие элементы:

триггер переполнения (один бит, который может принимать значение "нуль" или "единица");

флаг сравнения (принимающий одно из трех значений. LESS (меньше), EQUAL (равно) и GREATER (больше));

память (4 000 слов, каждое из которых состоит из 5 байт и знака);

устройства ввода-вывода (перфокарты, ленты, диски и т. д.).

Структура машинного слова. 5 байт и знак, из которых состоит машинное слово, нумеруются следующим образом:

0	1	2	3	4	5
±	Байт	Байт	Байт	Байт	Байт

(2)

Большинство команд таковы, что программист может при желании использовать только часть слова. В подобных случаях можно задать нестандартную "спецификацию поля". При этом допустимо использовать поля, которые являются соседними в машинном слове; они обозначаются в виде (L·R), где L — номер левой, а R — номер правой части поля. Приведем примеры спецификации полей:

(0:0): только знак;
(0:2): знак и первые два байта;
(0:5): целое слово; это самая распространенная спецификация поля;
(1:5): все слово, кроме знака;
(4:4): только четвертый байт;
(4:5): два младших значащих байта.

Использование спецификации поля несколько меняется от команды к команде; при рассмотрении каждой команды мы поговорим об этом более подробно. На самом деле каждая спецификация поля (L:R) представляется внутри компьютера одним числом — 8L + R; заметим, что это число легко помещается в одном байте

Формат команды. Машинные слова, используемые как команды, имеют следующий формат:

0	1	2	3	4	5
±	A	A	I	F	C

(3)

Крайний байт справа, C, — это *код операции*, который указывает, какая операция должна быть выполнена. Например, C = 8 определяет операцию LDA*, "загрузить регистр A".

Байт F определяет *модификацию* кода операции. Обычно это спецификация поля (L:R) = 8L+R. Например, если C = 8 и F = 11, то операцией будет "загрузить в регистр A поле (1:3)" Иногда F используется для других целей Например, для команд ввода-вывода F — это номер соответствующего входного или выходного устройства.

* LDA — сокращение от "load the A register" — *Прим. перев.*

Левая часть команды, ±AA, определяет *адрес*. (Обратите внимание, что знак является частью адреса.) Поле I, которое следует за адресом, — это *спецификация индекса*, которую можно применять для модификации фактического адреса. Если I = 0, то адрес ±AA используется без изменений. В противном случае в поле I должно содержаться число i от 1 до 6, и тогда содержимое индексного регистра Ii алгебраически добавляется к ±AA перед выполнением команды. Полученный результат используется как адрес. Процесс индексирования выполняется для *каждой* команды. Обозначим буквой "M" адрес, получаемый после каждой операции индексирования. (Если после добавления содержимого индексного регистра к адресу ±AA получается результат, который не помещается в двух байтах, то значение M будет неопределенным.)

Для большинства команд M указывает на ячейку памяти. Термины "ячейка памяти" и "адрес ячейки памяти" в этой книге почти всегда являются эквивалентными. Предполагается, что имеется 4 000 ячеек памяти с номерами от 0 до 3 999 Поэтому адрес каждой ячейки можно представить с помощью двух байтов. Для каждой команды, в которой M обозначает ячейку памяти, должно выполняться неравенство $0 \le M \le 3999$. В этом случае запись `CONTENTS(M)` будет обозначать величину, которая хранится в ячейке с адресом M.

Для некоторых команд "адрес" M имеет несколько иной смысл; он может даже быть отрицательным. Так, например, одна команда добавляет M к индексному регистру и при этом принимается во внимание знак M.

Форма записи. Чтобы сделать команды более читабельными, для обозначения команды типа (3) будет использоваться следующая форма записи:

```
OP    ADDRESS,I(F) .                    (4)
```

Здесь `OP` — символическое имя кода операции (часть C) команды, `ADDRESS` — часть ±AA, `I` и `F` — поля I и F соответственно.

Если I равно нулю, то запись "`,I`" опускается. Если F — *стандартная* F-спецификация для данной команды, то запись "`(F)`" использовать не нужно. Практически для всех команд стандартной F-спецификацией является (0:5), что соответствует целому слову. Если же стандартной является другая спецификация F, то она будет упомянута особо при описании конкретной команды

Например, команда загрузки числа в аккумулятор называется `LDA` и ее код операции — 8. Имеем следующее.

Условное представление	Реальное представление команды в цифровом виде				
`LDA 2000,2(0:3)`	+	2000	2	3	8
`LDA 2000,2(1:3)`	+	2000	2	11	8
`LDA 2000(1:3)`	+	2000	0	11	8
`LDA 2000`	+	2000	0	5	8
`LDA -2000,4`	-	2000	4	5	8

(5)

Команда "`LDA 2000,2(0:3)`" читается следующим образом: "Загрузить в регистр A содержимое ячейки 2000 с индексом 2, поле "нуль–три".

Для представлений цифрового содержимого слова MIX будем использовать клеточную запись, как было сделано выше. Обратите внимание, что в слове

+	2000	2	3	8

число +2000 занимает два соседних байта и знак. Заметим, что реальное содержимое байта (1:1) и байта (2:2) будет меняться при переходе от одного компьютера MIX к другому, так как меняется размер байта. В следующем примере подобной записи

-	10000	3000

представлено слово, состоящее из двух полей. В первом поле, содержащем три байта и знак, находится число −10000, а во втором поле, размером два байта, — число 3000. Когда слово разбито на несколько полей (т. е. содержит более одного поля), говорят, что оно "упаковано".

Описания команд. В замечаниях, следующих за записью (3) (см. выше), были определены величины M, F и C для любого слова, используемого в качестве команды. А теперь определим действия, соответствующие каждой команде.

Команды загрузки

• LDA (load A — загрузить A). C = 8; F = поле.
Заданное поле CONTENTS(M) заменяет предыдущее содержимое регистра A.

Во всех операциях, в которых в качестве входного значения используется частичное поле, знак учитывается, если он является частью этого поля; в противном случае берется знак "+". По мере загрузки поле сдвигается в правую часть регистра.

Примеры. Если F — стандартная спецификация поля (0:5), то все содержимое ячейки M копируется в rA. Если F — спецификация (1:5), то абсолютное значение CONTENTS(M) загружается со знаком "+". Если в M содержится *команда* и F — это спецификация (0:2), поле "±AA" загружается как

±	0	0	0	A	A

.

Предположим, в ячейке с адресом 2000 содержится слово

-	80	3	5	4

; (6)

тогда, загружая различные частичные поля, получим следующие результаты.

Команда		Последующее содержимое rA				
LDA	2000	-	80		3	5	4
LDA	2000(1:5)	+	80		3	5	4
LDA	2000(3:5)	+	0	0	3	5	4
LDA	2000(0:3)	-	0	0	80		3
LDA	2000(4:4)	+	0	0	0	0	5
LDA	2000(0:0)	-	0	0	0	0	0
LDA	2000(1:1)	+	0	0	0	0	?

(В последнем примере значение частично не определено, так как размер байта переменный.)

• LDX (load X — загрузить X). C = 15; F = поле.
Эта команда идентична LDA, за исключением того, что вместо rA загружается rX.

• LDi (load i — загрузить i). C = 8 + i; F = поле.
Эта команда идентична LDA, только вместо rA загружается r$I i$. Индексный регистр содержит только два байта (а не пять) и знак; байты 1, 2, 3 всегда считаются нулевыми. Поэтому, если установить для байтов 1, 2 или 3 любые значения. не равные нулю, то команда LDi станет неопределенной.

В описаниях всех команд "i" обозначает целое число, $1 \le i \le 6$. Таким образом. LDi обозначает шесть различных команд: LD1, LD2, ..., LD6.

• LDAN (load A negative — загрузить в A с обратным знаком). C = 16; F = поле.
• LDXN (load X negative — загрузить в X с обратным знаком). C = 23; F = поле.
• LDiN (load i negative — загрузить в i с обратным знаком). C = 16 + i; F = поле.
Эти восемь команд идентичны командам LDA, LDX и LDi соответственно. но только величины загружаются с *обратным* знаком.

Команды записи в память

• STA (store A — записать A). C = 24; F = поле.
Часть содержимого rA заменяет поле CONTENTS(M), которое указано в F Другие части CONTENTS(M) остаются неизменными.

Для операции *записи в память* поле F имеет противоположный смысл по сравнению с операцией *загрузки*. Нужное количество байтов в поле, взятом из правой части регистра, в случае необходимости сдвигается *влево*, а затем помещается в соответствующее поле CONTENTS(M). Знак меняется только тогда, когда он является частью поля. Содержимое регистра также остается без изменений.

Примеры. Предположим, в ячейке 2000 содержится

-	1	2	3	4	5

а в регистре A -

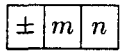
+	6	7	8	9	0

Тогда получаем следующее.

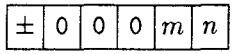
Команда	Содержимое ячейки 2000 после выполнения команды					
STA 2000	+	6	7	8	9	0
STA 2000(1:5)	-	6	7	8	9	0
STA 2000(5:5)	-	1	2	3	4	0
STA 2000(2:2)	-	1	0	3	4	5
STA 2000(2:3)	-	1	9	0	4	5
STA 2000(0:1)	+	0	2	3	4	5

• STX (store X — записать X) C = 31: F = поле.
Идентична команде STA, но только сохраняется содержимое rX, а не rA.

• STi (store i — записать i). C = 24 + i; F = поле.
Идентична команде STA, за исключением того, что сохраняется содержимое rIi, а не rA. Байты 1, 2 и 3 индексного регистра являются нулевыми, поэтому, если в rI1 содержится



значит, в нем находится



• STJ (store J — сохранить J). C = 32; F = поле.
Идентична команде STi, но только сохраняется rJ и знаком всегда является "+".

Для команды STJ стандартной спецификацией поля для F является (0:2), *а не* (0:5). И это естественно, так как STJ почти всегда выполняет запись в адресное поле команды.

• STZ (store zero — записать нуль). C = 33; F = поле.
Идентична команде STA, но только сохраняется нуль со знаком "+". Другими словами, заданное поле CONTENTS(M) обнуляется.

Арифметические команды. При выполнении операций сложения, вычитания, умножения и деления допускается спецификация поля. Спецификацию поля "(0:6)" можно использовать для указания операции с плавающей точкой (см. раздел 4.2). Но лишь в некоторых программах для MIX мы будем пользоваться этой возможностью, так как, в первую очередь, нас будут интересовать целочисленные алгоритмы.

Стандартной спецификацией поля, как обычно, является (0:5). Остальные поля имеют такой же смысл, как при выполнении команды LDA. Буквой "V" будем обозначать заданное поле CONTENTS(M). Таким образом, V — это значение, которое должно быть загружено в регистр A, если кодом операции является LDA.

• ADD (сложение). C = 1; F = поле.
V добавляется к rA. Если абсолютное значение результата слишком велико для того, чтобы поместиться в регистре A, то для триггера переполнения устанавливается значение 1, а в rA остается значение, которое выглядит так, как будто "1" перенесено в другой регистр слева от rA. (В противном случае значение для триггера переполнения не меняется.) Если результат равен нулю, то знак регистра rA остается неизменным.

Пример. Последовательность команд, приведенных ниже, служит для вычисления суммы значений, находящихся в пяти байтах регистра A.

```
STA  2000
LDA  2000(5:5)
ADD  2000(4:4)
ADD  2000(3:3)
ADD  2000(2:2)
ADD  2000(1:1)
```

Иногда эту операцию называют сложением наискосок.

В одних компьютерах MIX переполнение происходит, а в других — нет, что связано с разницей в размерах байта. Мы не говорили, что переполнение обязательно произойдет, если результат превысит число 1 073 741 823; переполнение случается,

когда абсолютное значение результата не помещается в пяти байтах (в зависимости от размера байта). Тем не менее можно написать программы, которые правильно работают и дают одинаковые окончательные результаты независимо от размера байта.

• SUB (вычитание). C = 2; F = поле.
V вычитается из rA. (Эквивалентна команде ADD, только вместо V берется −V.)

• MUL (умножение). C = 3; F = поле.
Занимающее 10 байт произведение V и значения из rA заменяет содержимое регистров A и X. В качестве знаков rA и rX устанавливается алгебраический знак произведения (а именно — "+", если знаки V и rA были одинаковы, и "−", если они были различны).

• DIV (деление). C = 4; F = поле.
Значение из rA и rX, рассматриваемое как 10-байтовое число rAX со знаком rA, делится на значение V. Если V = 0 или если абсолютная величина частного не помещается в пяти байтах (это эквивалентно условию $|\mathrm{rA}| \geq |\mathrm{V}|$), то регистры A и X заполняются неопределенной информацией и для триггера переполнения устанавливается значение 1. В противном случае частное $\pm\lfloor |\mathrm{rAX}/\mathrm{V}| \rfloor$ помещается в rA, а остаток $\pm(|\mathrm{rAX}| \bmod |\mathrm{V}|)$ — в rX. После выполнения операции знаком rA становится алгебраический знак частного (а именно — "+", если знаки V и rA были одинаковы, и "−", если они были различны). Знаком rX после выполнения операции будет тот знак, который был у rA до ее выполнения.

Примеры арифметических команд. В большинстве случаев арифметические действия выполняются только со словами MIX, которые являются одинарными пятибайтовыми числами, не упакованными в нескольких полях. Тем не менее можно выполнять арифметические операции и над упакованными словами MIX, если принять некоторые меры предосторожности. Для этого следует внимательно изучить приведенные ниже примеры. (Как и раньше, ? обозначает неопределенное значение.)

	+	1234	1	150		rA до операции
	+	100	5	50		Ячейка 1000
ADD 1000	+	1334	6	200		rA после операции

	−	1234	0	0	9	rA до операции
	−	2000	150		0	Ячейка 1000
SUB 1000	+	766	149		?	rA после операции

	+	1	1	1	1	1	rA до операции
	+	1	1	1	1	1	Ячейка 1000
MUL 1000	+	0	1	2	3	4	rA после операции
	+	5	4	3	2	1	rX после операции

	-				112	rA до операции
	?	2	?	?	?	?	Ячейка 1000
MUL 1000(1:1)	-				0	rA после операции
	-				224	rX после операции

	-	50	0	112	4	rA до операции
	-	2	0	0	0	0	Ячейка 1000
MUL 1000	+	100	0	224	rA после операции
	+	8	0	0	0	0	rX после операции

	+				0	rA до операции
	?				17	rX до операции
	+				3	Ячейка 1000
DIV 1000	+				5	rA после операции
	+				2	rX после операции

	-				0	rA до операции
	+	1235	0	3	1	rX до операции
	-	0	0	0	2	0	Ячейка 1000
DIV 1000	+	0	617	?	?	rA после операции
	-	0	0	0	?	1	rX после операции

(Эти примеры были подготовлены с учетом той точки зрения, что лучше дать трудное, но полное описание, чем простое, но неполное.)

Команды операций с адресами. В следующих операциях "адрес" M (возможно, индексированный) используется как число со знаком, а не как адрес ячейки памяти.

• `ENTA` (enter A). C = 48; F = 2.
Величина M загружается в rA. Это действие эквивалентно команде "`LDA`" для загрузки из памяти слова, содержащего число M со знаком. Если M = 0, то загружается знак команды.

Примеры. "`ENTA 0`" обнуляет rA и устанавливает для него знак "+". "`ENTA 0,1`" заносит в rA текущее содержимое индексного регистра 1, только −0 меняется на +0. Действие команды "`ENTA -0,1`" аналогично, только +0 меняется на −0.

• `ENTX` (enter X — ввести X). C = 55; F = 2.
• `ENT`i (enter i — ввести i). C = 48 + i; F = 2.
Эти команды аналогичны `ENTA`, но только загружается соответствующий регистр.

• `ENNA` (enter negative A — ввести A с обратным знаком). C = 48; F = 3.
• `ENNX` (enter negative X — ввести X с обратным знаком). C = 55; F = 3.

• ENNi (enter negative i — ввести i с обратным знаком). C = 48 + i; F = 3.
Эти команды идентичны ENTA, ENTX и ENTi, но в данном случае при загрузке знак меняется на противоположный.

Пример. Команда "ENN3 0,3" меняет на противоположный знак содержимого регистра rI3, хотя −0 так и остается −0.

• INCA (increase A — увеличить A). C = 48; F = 0.
К содержимому регистра rA добавляется величина M; это эквивалент команды ADD для добавления из памяти слова, содержащего величину M. Здесь также возможно переполнение, которое обрабатывается, как и в случае команды ADD.

Пример. Команда "INCA 1" увеличивает содержимое rA на единицу.

• INCX (increase X — увеличить X). C = 55; F = 0.
Величина M добавляется к содержимому rX. Если происходит переполнение, то оно обрабатывается так же, как в случае команды ADD, только вместо rA используется rX. Действие этой команды никогда не затрагивает регистр A.

• INCi (increase i — увеличить i). $C = 48 + i$; F = 0.
К содержимому rIi добавляется величина M. Переполнения быть не должно; если M + rIi не помещается в двух байтах, то результат команды считается неопределенным.

• DECA (decrease A — уменьшить A). C = 48; F = 1.
• DECX (decrease X — уменьшить X). C = 55; F = 1.
• DECi (decrease i — уменьшить i). C = 48 + i; F = 1.
Эти восемь команд аналогичны INCA, INCX и INCi соответственно, но только M не добавляется к содержимому регистра, а вычитается из него.

Обратите внимание, что для команд ENTA, ENNA, INCA и DECA используется один и тот же код операции C; поле F используется для того, чтобы можно было отличить одну операцию от другой.

Команды сравнения. При выполнении всех команд сравнения MIX сравнивается величина, содержащаяся в регистре, с величиной, содержащейся в памяти. Затем для флага сравнения устанавливается значение LESS (меньше), EQUAL (равно) или GREATER (больше) в зависимости от того, будет ли содержащееся в регистре значение *меньше, равно или больше величины, содержащейся в ячейке памяти.* При этом нуль со знаком "−" считается *равным* нулю со знаком "+".

• CMPA (compare A — сравнить A). C = 56; F = поле.
Заданное поле rA сравнивается с *тем же* полем CONTENTS(M). Если F не содержит бит знака, то оба поля считаются неотрицательными; в противном случае сравнение выполняется с учетом знака. (Равенство всегда будет результатом сравнения, если F — это (0:0), так как нуль со знаком "−" равен нулю со знаком "+".)

• CMPX (compare X — сравнить X). C = 63; F = поле.
Эта команда аналогична CMPA.

• CMPi (compare i — сравнить i). C = 56 + i; F = *поле*.
Аналог CMPA. Байты 1, 2 и 3 индексного регистра при сравнении считаются нулевыми. (Поэтому, если F = (1:2), результатом сравнения не может быть знак GREATER (больше).)

Команды перехода. Команды обычно выполняются последовательно. Другими словами, команда, которая выполняется после команды из ячейки P, обычно находится в ячейке P + 1. Но команды "перехода" позволяют нарушить этот последовательный ход выполнения. При выполнении типичной команды перехода в регистр J заносится адрес следующей команды (т. е. команды, которая оказалась бы следующей, не будь перехода). Затем в случае необходимости программист сможет использовать "адрес J" для определения адресного поля другой команды, чтобы вернуться к первоначальному месту программы. Содержимое регистра J меняется при каждом переходе в программе, за исключением случая, когда используется команда перехода JSJ. Если перехода нет, содержимое этого регистра измениться никак не может.

• JMP (jump — перейти). C = 39; F = 0.
Команда безусловного перехода: следующая команда выбирается из ячейки M.

• JSJ (jump, save J — перейти, сохранить J). C = 39; F = 1.
Эта команда идентична JMP, но только содержимое rJ не меняется.

• JOV (jump on overflow — перейти при переполнении). C = 39; F = 2.
Если для флага переполнения установлено значение 1, то он переключается в положение 0 и выполняется команда JMP; в противном случае ничего не происходит.

• JNOV (jump on no overflow — перейти, если нет переполнения). C = 39; F = 3.
Если для флага переполнения установлено значение 0, то выполняется команда JMP; в противном случае ничего не происходит.

• JL, JE, JG, JGE, JNE, JLE (jump on less, equal, greater, greater-or-equal, unequal, less-or-equal — перейти, если меньше, равно, больше, больше или равно, не равно, меньше или равно). C = 39; F = 4, 5, 6, 7, 8, 9 соответственно.
Переход осуществляется в случае, если для флага сравнения установлено указанное значение. Например, по команде JNE переход будет выполнен, если значением флага сравнения является LESS (меньше) или GREATER (больше). Заметим, что эти команды не изменяют значение самого флага сравнения.

• JAN, JAZ, JAP, JANN, JANZ, JANP (jump A negative, zero, positive, nonnegative, nonzero, nonpositive — перейти, если в регистре A отрицательное значение, нулевое, положительное, ненулевое, неположительное). C = 40; F = 0, 1, 2, 3, 4, 5 соответственно.
Если содержимое rA удовлетворяет заданному условию, то выполняется команда JMP, в противном случае ничего не происходит. "Положительным" является значение, которое *больше* нуля (но не нуль), а "неположительным" — наоборот, нуль или отрицательное значение.

• JXN, JXZ, JXP, JXNN, JXNZ, JXNP (jump X negative, zero, positive, nonnegative, nonzero, nonpositive — перейти, если в регистре X отрицательное значение, нуль, положительное, неотрицательное, ненулевое, неположительное). C = 47; F = 0, 1, 2, 3, 4, 5 соответственно.

• J*i*N, J*i*Z, J*i*P, J*i*NN, J*i*NZ, J*i*NP (jump i negative, zero, positive, nonnegative, nonzero, nonpositive — перейти, если в индексном регистре i — отрицательное значение, нуль, положительное, неотрицательное, ненулевое, неположительное). C = 40 + i; F = 0, 1, 2, 3, 4, 5 соответственно. Это аналоги соответствующих команд для rA.

Другие команды

• SLA, SRA, SLAX, SRAX, SLC, SRC (shift left A (сдвинуть A влево), shift right A (сдвинуть A вправо), shift left AX (сдвинуть AX влево), shift right AX (сдвинуть AX вправо), shift left AX circularly (циклический сдвиг AX влево), shift right AX circularly (циклический сдвиг AX вправо)). C = 6; F = 0, 1, 2, 3, 4, 5 соответственно.

Это команды "сдвига", где M обозначает число байтов компьютера MIX, которые нужно сдвинуть вправо или влево; M должно быть неотрицательным. Команды SLA и SRA не оказывают влияния на содержимое rX; остальные команды сдвига оказывают такое действие на оба регистра A и X, как будто это один 10-байтовый регистр. При выполнении команд SLA, SRA, SLAX и SRAX в регистр с одной стороны входят нули, а с другой стороны исчезает информация из сдвинутых байтов. Команды SLC и SRC вызывают "циклический" сдвиг, при котором байты, "исчезнувшие" с одной стороны, снова входят в регистр с другой стороны. В циклическом сдвиге участвуют оба регистра — rA и rX. Заметим, что ни одна из команд сдвига никак не влияет на знаки регистров A и X.

Примеры

	Регистр A						Регистр X					
Первоначальное содержимое	+	1	2	3	4	5	-	6	7	8	9	10
SRAX 1	+	0	1	2	3	4	-	5	6	7	8	9
SLA 2	+	2	3	4	0	0	-	5	6	7	8	9
SRC 4	+	6	7	8	9	2	-	3	4	0	0	5
SRA 2	+	0	0	6	7	8	-	3	4	0	0	5
SLC 1	+	0	6	7	8	3	-	4	0	0	5	0

• MOVE (переместить). C = 7; F = число.

Количество слов, определенное значением F, перемещается, начиная от ячейки M, в другие ячейки, адрес первой из которых задается содержимым индексного регистра 1. Перемещение осуществляется по одному слову за раз, и к концу выполнения операции значение в регистре rI1 увеличивается на F. Если F = 0, то ничего не происходит.

Необходимо следить за тем, чтобы группы ячеек, участвующих в перемещении, не перекрывались. Предположим, что F = 3 и M = 1000. Тогда, если rI1 = 999, перемещаем CONTENTS(1000) в CONTENTS(999), CONTENTS(1001) в CONTENTS(1000) и CONTENTS(1002) в CONTENTS(1001); в данном случае все нормально. Но если бы в регистре rI1 содержалось число 1001, то в результате были бы выполнены перемещения CONTENTS(1000) в CONTENTS(1001), CONTENTS(1001) в CONTENTS(1002), CONTENTS(1002) в CONTENTS(1003), т. е. мы бы переместили *одно и то же* слово CONTENTS(1000) в три различных места.

• NOP (no operation — нет операции). C = 0.

Никакие действия не выполняются, и эта команда просто пропускается. F и M игнорируются.

• HLT (halt — остановить). C = 5; F = 2.

Остановка работы компьютера. Пока оператор будет его перезапускать, все будет выглядеть так, как будто работает команда NOP (т. е. никакие действия не выполняются).

Команды ввода-вывода. MIX можно оснастить достаточно большим количеством устройств ввода-вывода (причем все они поставляются за дополнительную плату). Каждому устройству соответствует определенный номер.

Номер устройства	Периферийное устройство	Размер блока, слов
t	Накопитель на магнитной ленте номер t $(0 \le t \le 7)$	100
d	Диск или барабан номер d $(8 \le d \le 15)$	100
16	Устройство чтения перфокарт	16
17	Перфоратор	16
18	Принтер	24
19	Терминал ввода данных	14
20	Перфолента	14

Не каждый компьютер MIX будет оснащен всеми этими устройствами, поэтому время от времени мы будем специально упоминать о наличии тех или иных устройств. Некоторые из них нельзя использовать и для ввода, и для вывода. В приведенной выше таблице указаны фиксированные размеры блоков (в словах) для каждого устройства.

При вводе или выводе с помощью таких устройств, как накопитель на магнитной ленте, диск или барабан, происходит чтение или запись полных слов (состоящих из пяти байтов и знака). Но устройства ввода-вывода с номерами от 16 до 20 всегда работают в *символьном коде*, в котором каждый байт представляет один буквенно-цифровой символ. Поэтому с помощью каждого слова MIX передается сразу пять символов. Символьный код приведен в верхней части табл. 1, которая находится в конце данного раздела, и на форзацах в конце книги. Код 00 соответствует символу "␣", который обозначает *пробел*. Коды 01–29 представляют буквы от A до Z, среди которых есть несколько греческих букв; коды 30–39 представляют цифры 0, 1, ..., 9, а следующие коды 40, 41, ... — знаки пунктуации и другие специальные символы. (Набор символов MIX отражает положение дел на то время, когда компьютеры еще не могли справиться со строчными буквами.) С помощью символьного кода нельзя прочитать или записать все возможные величины, которые могут содержаться в байте, так как некоторые комбинации не определены. Более того, некоторые устройства ввода-вывода могут оказаться не предназначенными для обработки всех элементов из набора символов, например символы Σ и Π, встречающиеся в тексте, скорее всего, не будут восприняты устройством чтения перфокарт. Когда данные вводятся в символьном коде, всем словам присваиваются знаки "+", а при выводе знаки игнорируются. Если данные вводятся с терминала, то набор в конце каждой строки символа возврата каретки приводит к тому, что остаток строки заполняется пробелами.

Дисковые и барабанные устройства — это внешние устройства памяти, каждое из которых содержит блоки из 100 слов. При выполнении каждой команды IN, OUT или IOC (см. ниже) конкретный блок из 100 слов, на который ссылается команда, определяется текущим содержимым rX и не должен превосходить емкость используемого диска или барабана.

• IN (input — ввод). C = 36; F = номер устройства.
Эта команда реализует передачу информации из заданного устройства ввода в последовательно расположенные ячейки, начиная с ячейки M. Число ячеек, из которых передается информация, соответствует размеру блока для данного устройства (см. приведенную выше таблицу). Если предыдущая операция на этом же устройстве еще не закончена, то компьютер будет ожидать ее завершения. Время, в течение которого будет длиться передача информации, начатая по этой команде, зависит от скорости работы устройства ввода. Поэтому до момента завершения передачи информации программа не должна обращаться к этой информации в памяти. Не следует пытаться прочитать с магнитной ленты любой блок, следующий за блоком, который был записан последним.

• OUT (output — вывод). C = 37; F = номер устройства.
Эта команда реализует передачу информации из ячеек памяти, начиная с ячейки M, на заданное устройство вывода. Если сначала устройство не было готово, то компьютер будет ждать его готовности. Время, в течение которого будет длиться передача информации, начатая по этой команде, зависит от скорости устройства вывода. Поэтому до момента завершения передачи информации программа не должна производить изменения в соответствующих ячейках памяти.

• IOC (input-output control — управление вводом-выводом). C = 35; F = номер устройства.
В случае необходимости компьютер ожидает, пока освободится заданное устройство. Затем выполняется управляющая команда, которая зависит от типа применяемого устройства. В этой книге будут использоваться следующие примеры.

Магнитная лента. Если M = 0, то лента перематывается в начало. Если M < 0, то лента перематывается на −M блоков (т. е. на M блоков назад) или в начало, в зависимости от того, что произойдет раньше. Если M > 0, то лента перематывается вперед; перематывая ленту вперед, нельзя заходить дальше блока, который был записан последним.

Например, последовательность команд "OUT 1000(3): IOC -1(3); IN 2000(3)" записывает сто слов на ленту 3, а затем снова считывает их. Если надежность ленты не ставится под сомнение, то использование последних двух команд этой последовательности представляет собой медленный способ перемещения слов 1000–1099 в ячейки 2000–2099. Последовательность команд "OUT 1000(3); IOC +1(3)" некорректна.

Диск или барабан. M должно быть равно нулю. В результате устройство позиционируется в соответствии с содержимым rX, чтобы следующая операция IN или OUT на этом устройстве выполнялась быстрее в случае, если используется то же значение rX.

Принтер. M должно быть равно нулю. Команда "IOC 0(18)" заставит принтер перейти к началу (т. е. к верху) следующей страницы.

Перфолента. M должно быть равно нулю. Команда "IOC 0(20)" перематывает ленту в начало.

• JRED (jump ready — переход при готовности). C = 38; F = номер устройства.
Переход происходит в случае, если заданное устройство готово, т. е. завершена предыдущая операция, инициированная командой IN, OUT или IOC.

• JBUS (jump busy — переход при занятости). C = 34; F = номер устройства.
Антипод команды JRED: переход происходит, если заданное устройство *не* готово.

Пример. Команда "JBUS 1000(16)" из ячейки 1000 будет повторно выполняться до тех пор, пока устройство 16 *не* будет готово.

Перечисленные выше простые команды исчерпывают набор команд ввода-вывода компьютера MIX. В этом компьютере нет индикаторов контроля ленты и т. д., служащих для предотвращения аварийных ситуаций на периферийных устройствах. Любая подобная ситуация (бумага застревает, устройство отключается, лента отсутствует и т. д.) приводит к тому, что устройство остается занятым, звенит звонок и опытный оператор устраняет проблемы вручную, выполняя обычные процедуры обслуживания. О более сложных периферийных устройствах, которые являются более дорогими и более типичными представителями современного оборудования по сравнению с описанными здесь лентами, барабанами и дисками с фиксированным размером блоков, пойдет речь в разделах 5.4.6 и 5.4.9.

Команды преобразования

• NUM (convert to numeric — преобразовать в число). C = 5; F = 0.
Эта команда используется для преобразования символьного кода в число. M игнорируется. Предполагается, что регистры A и X содержат 10-байтовое число в символьном коде и команда NUM заносит в rA полученное числовое значение (которое рассматривается как десятичное число). Содержимое rX и знак rA не меняются. Байты 00, 10, 20, 30, 40, ... преобразуются в нулевые; в байты 01, 11, 21, ... заносится цифра 1 и т. д. Если происходит переполнение (что вполне возможно), сохраняется остаток по модулю b^5, где b — размер байта.

• CHAR (convert to characters — преобразовать в символы). C = 5; F = 1.
Эта операция используется для преобразования машинного кода в символьный код, который воспринимается устройством вывода на перфокарты, ленту или на принтер. Значение из rA преобразуется в 10-байтовое десятичное число, которое заносится в регистры A и X в символьном коде. Знаки rA и rX не меняются. M игнорируется.

Примеры

	Регистр A						Регистр X					
Первоначальное содержимое	-	00	00	31	32	39	+	37	57	47	30	30
NUM 0	-				12977700		+	37	57	47	30	30
INCA 1	-				12977699		+	37	57	47	30	30
CHAR 0	-	30	30	31	32	39	+	37	37	36	39	39

Время выполнения. Чтобы определить количественные характеристики эффективности программ для MIX, для каждой его команды задано *время выполнения*, которое типично для компьютеров "урожая 1970 года".

ADD, SUB, все команды LOAD, все команды STORE (включая STZ), все команды сдвига и все команды сравнения выполняются в течение *двух тактов*. Для выполнения команды MOVE требуется один такт плюс еще два на каждое перемещаемое слово. Для каждой команды MUL, NUM, CHAR требуется 10 тактов, а для DIV — 12. Время выполнения операций с плавающей точкой определяется в разделе 4.2.1. Для всех остальных операций необходим один такт плюс время, в течение которого компьютер может ожидать завершения выполнения команд IN, OUT, IOC и HLT.

Таблица 1

Код символа:

00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
␣	A	B	C	D	E	F	G	H	I	Δ	J	K	L	M	N	O	P	Q	R	Σ	Π	S	T	U

00	*1*	**01**	*2*	**02**	*2*	**03**	*10*

Нет операции
`NOP(0)`
		rA ← rA + V
		`ADD(0:5)`
		`FADD(6)`
				rA ← rA − V
				`SUB(0:5)`
				`FSUB(6)`
						rAX ← rA × V
						`MUL(0:5)`
						`FMUL(6)`	

08	*2*	**09**	*2*	**10**	*2*	**11**	*2*
rA ← V `LDA(0:5)`		rI1 ← V `LD1(0:5)`		rI2 ← V `LD2(0:5)`		rI3 ← V `LD3(0:5)`	
16	*2*	**17**	*2*	**18**	*2*	**19**	*2*
rA ← −V `LDAN(0:5)`		rI1 ← −V `LD1N(0:5)`		rI2 ← −V `LD2N(0:5)`		rI3 ← −V `LD3N(0:5)`	
24	*2*	**25**	*2*	**26**	*2*	**27**	*2*
M(F) ← rA `STA(0:5)`		M(F) ← rI1 `ST1(0:5)`		M(F) ← rI2 `ST2(0:5)`		M(F) ← rI3 `ST3(0:5)`	
32	*2*	**33**	*2*	**34**	*1*	**35**	*1+T*
M(F) ← rJ `STJ(0:2)`		M(F) ← 0 `STZ(0:5)`		Устр. F занято? `JBUS(0)`		Упр. устр. F `IOC(0)`	
40	*1*	**41**	*1*	**42**	*1*	**43**	*1*
rA : 0, переход `JA[+]`		rI1 : 0, переход `J1[+]`		rI2 : 0, переход `J2[+]`		rI3 : 0, переход `J3[+]`	
48	*1*	**49**	*1*	**50**	*1*	**51**	*1*

rA ← [rA]? ± M
`INCA(0) DECA(1)`
`ENTA(2) ENNA(3)`
		rI1 ← [rI1]? ± M
		`INC1(0) DEC1(1)`
		`ENT1(2) ENN1(3)`
				rI2 ← [rI2]? ± M
				`INC2(0) DEC2(1)`
				`ENT2(2) ENN2(3)`
						rI3 ← [rI3]? ± M
						`INC3(0) DEC3(1)`
						`ENT3(2) ENN3(3)`	

56	*2*	**57**	*2*	**58**	*2*	**59**	*2*

CI ← rA(F) : V
`CMPA(0:5)`
`FCMP(6)`
		CI ← rI1(F) : V
		`CMP1(0:5)`
				CI ← rI2(F) : V
				`CMP2(0:5)`
						CI ← rI3(F) : V
						`CMP3(0:5)`	

Общая форма записи:

C	*t*
Описание `OP(F)`	

C — код операции, поле команды (5:5)
F — уточн. кода опер., поле команды (4:4)
M — адрес команды после индексации
V = M(F) — содержимое поля F ячейки M
OP — символический код операции
(F) — стандартное значение F
t — время выполнения; *T* — время блокировки

25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55
V	W	X	Y	Z	0	1	2	3	4	5	6	7	8	9	.	,	(	)	+	-	*	/	=	$	<	>	@	;	:	'

04	*12*	**05**	*10*	**06**	*2*	**07**	*1 + 2F*
rA ← rAX/V rX ← остаток DIV(0:5) FDIV(6)		Специальные NUM(0) CHAR(1) HLT(2)		Сдвиг на M байт SLA(0) SRA(1) SLAX(2) SRAX(3) SLC(4) SRC(5)		Переместить F слов из M в rI1 MOVE(1)	
12	*2*	**13**	*2*	**14**	*2*	**15**	*2*
rI4 ← V LD4(0:5)		rI5 ← V LD5(0:5)		rI6 ← V LD6(0:5)		rX ← V LDX(0:5)	
20	*2*	**21**	*2*	**22**	*2*	**23**	*2*
rI4 ← −V LD4N(0:5)		rI5 ← −V LD5N(0:5)		rI6 ← −V LD6N(0:5)		rX ← −V LDXN(0:5)	
28	*2*	**29**	*2*	**30**	*2*	**31**	*2*
M(F) ← rI4 ST4(0:5)		M(F) ← rI5 ST5(0:5)		M(F) ← rI6 ST6(0:5)		M(F) ← rX STX(0:5)	
36	*1 + T*	**37**	*1 + T*	**38**	*1*	**39**	*1*
Ввод с устр. F IN(0)		Вывод на устр. F OUT(0)		Устр. F готово? JRED(0)		Переходы JMP(0) JSJ(1), JOV(2) JNOV(3), а также [*] ниже	
44	*1*	**45**	*1*	**46**	*1*	**47**	*1*
rI4 : 0, переход J4[+]		rI5 : 0, переход J5[+]		rI6 : 0, переход J6[+]		rX : 0, переход JX[+]	
52	*1*	**53**	*1*	**54**	*1*	**55**	*1*
rI4 ← [rI4]? ± M INC4(0) DEC4(1) ENT4(2) ENN4(3)		rI5 ← [rI5]? ± M INC5(0) DEC5(1) ENT5(2) ENN5(3)		rI6 ← [rI6]? ± M INC6(0) DEC6(1) ENT6(2) ENN6(3)		rX ← [rX]? ± M INCX(0) DECX(1) ENTX(2) ENNX(3)	
60	*2*	**61**	*2*	**62**	*2*	**63**	*2*
CI ← rI4(F) : V CMP4(0:5)		CI ← rI5(F) : V CMP5(0:5)		CI ← rI6(F) : V CMP6(0:5)		CI ← rX(F) : V CMPX(0:5)	

rA — регистр A
rX — регистр X
rAX — регистры A и X вместе
rIi — индексный регистр i, $1 \le i \le 6$
rJ — регистр J
CI — индикатор сравнения

[*]:		[+]:
JL(4)	<	N(0)
JE(5)	=	Z(1)
JG(6)	>	P(2)
JGE(7)	≥	NN(3)
JNE(8)	≠	NZ(4)
JLE(9)	≤	NP(5)

В частности, заметим, что ENTA выполняется в течение одной единицы времени, а LDA — в течение двух. Время выполнения команд легко запомнить, потому что, за исключением команд сдвига, преобразования, MUL и DIV, число единиц времени равно числу обращений к оперативной памяти (включая обращение к самой команде).

Основная единица измерения времени MIX — это относительная мера, которую мы просто обозначим через u. Ее можно считать равной, скажем, 10 мкс (для сравнительно недорогого компьютера) или 10 нс (для достаточно дорогого).

Пример. Последовательность команд LDA 1000; INCA 1; STA 1000 выполняется ровно $5u$.

Мой взор проник в ее глубины —
Туда, где бьется пульс машины.

— ВИЛЬЯМ ВОРДСВОРС (WILLIAM WORDSWORTH),
She Was a Phantom of Delight (1804)

Резюме. Итак, мы обсудили все характеристики MIX, за исключением его "кнопки GO" (пуск), о которой пойдет речь в упр. 26. Хотя в компьютере MIX приблизительно 150 различных операций, они вписываются в несколько простых схем и поэтому легко запоминаются. В табл. 1 приведены операции для параметра C. После имени каждой команды в круглых скобках указано стандартное значение поля F.

Следующие упражнения помогут лучше усвоить материал данного раздела. В основном, эти упражнения очень просты, так что постарайтесь их выполнить.

УПРАЖНЕНИЯ

1. [*00*] Если бы MIX был троичным компьютером, то сколько троичных чисел поместилось бы в байте?

2. [*02*] Если бы величина, которую нужно представить в машине MIX, могла достигать такого большого значения, как 99 999 999, то сколько соседних байтов было бы занято этой величиной?

3. [*02*] Укажите частичные спецификации поля (L:R) для (a) адресного поля, (b) индексного поля, (c) поля спецификации поля и (d) поля кода операций команды MIX.

4. [*00*] Последним примером в (5) является команда "LDA -2000,4". Насколько она законна ввиду того, что адреса памяти не должны быть отрицательными?

5. [*10*] Какая символическая запись, аналогичная (4), соответствует (6), если (6) рассматривать как команду MIX?

▸ **6.** [*10*] Предположим, что в ячейке 3000 содержится

+	5	1	200	15

.

Каким будет результат выполнения следующих команд? (Установите, являются ли какие-либо из них неопределенными или только частично определенными.) (a) LDAN 3000; (b) LD2N 3000(3:4); (c) LDX 3000(1:3); (d) LD6 3000; (e) LDXN 3000(0:0).

7. [*M15*] С помощью алгебраических операций $X \bmod Y$ и $\lfloor X/Y \rfloor$ дайте точное определение результатов выполнения команды DIV для всех случаев, когда не происходит переполнение.

8. [*15*] В *последнем примере команды* DIV на с. 165 *содержимым* rX *до операции является* | + | 1235 | 0 | 3 | 1 |. Если бы содержимым регистра было | - | 1234 | 0 | 3 | 1 |, в то время как все остальные условия примера остались бы прежними, то что содержалось бы в регистрах A и X после выполнения команды DIV?

▸ **9.** [*15*] Перечислите все команды MIX, которые могут изменить значение флага переполнения. (Исключите из рассмотрения операции с плавающей точкой.)

10. [*15*] Перечислите все команды MIX, которые могут изменить значение флага сравнения

▸ **11.** [*15*] Перечислите все команды MIX, которые могут изменить содержимое rI1.

12. [*10*] Найдите единственную команду, действие которой эквивалентно умножению текущего содержимого rI3 на два и сохранению результата в rI3.

▸ **13.** [*10*] Предположим, в ячейке 1000 содержится команда "JOV 1001" Эта команда переключает флаг переполнения в положение 0, если он находится в положении 1 (и в любом случае следующая команда выбирается из ячейки 1001). Изменится ли что-нибудь, если заменить эту команду командой "JNOV 1001"? Что будет, если заменить ее командой "JOV 1000" или "JNOV 1000"?

14. [*20*] Для каждой команды MIX выясните, существует ли способ определения части ±AA, I и F таким образом, чтобы она была эквивалентна NOP (за исключением того, что время выполнения может увеличиться). Предположим, о содержимом регистров либо ячеек памяти ничего не известно В тех случаях, когда возможно осуществить NOP, объясните, как это сделать. *Примеры.* INCA эквивалентна NOP, когда адресное и индексное поля равны нулю. JMP никогда не будет эквивалентна NOP, так как она изменяет содержимое rJ.

15. [*10*] Сколько *буквенно-цифровых символов* содержится в блоке данных терминала или устройства вывода на перфоленту; устройства чтения перфокарт или перфоратора; АЦПУ?

16. [*20*] Напишите программу, которая обнуляет все ячейки памяти 0000–0099 и является (a) настолько короткой, насколько это возможно; (b) настолько быстрой, насколько это возможно. [*Указание.* Воспользуйтесь командой MOVE.]

17. [*26*] Пусть выполнены условия предыдущего упражнения, только нужно обнулить ячейки с 0000 по N включительно, где N — текущее содержимое rI2. Ваши программы должны удовлетворять условиям (a) и (b) из предыдущего упражнения, работать для любого значения $0 \le N \le 2999$ и начинаться с ячейки 3000.

▸ **18.** [*22*] После того как будет выполнена следующая программа "номер один", какие изменения произойдут с регистрами, флагом переполнения, флагом сравнения и оперативной памятью? (Например, каким будет окончательное значение в регистре rI1 или rX? Каким будет значение флага переполнения и флага сравнения?)

```
STZ  1
ENNX 1
STX  1(0:1)
SLAX 1
ENNA 1
INCX 1
ENT1 1
SRC  1
ADD  1
DEC1 -1
STZ  1
CMPA 1
```

```
MOVE -1,1(1)
NUM  1
CHAR 1
HLT  1
```

▸ **19.** [*14*] Каким будет время выполнения программы из предыдущего примера без учета команды HLT?

20. [*20*] Напишите программу, которая заносит во *все* 4 000 ячеек памяти команду HLT, а затем останавливается.

▸ **21.** [*24*] (a) Может ли нуль вообще содержаться в регистре J? (b) Напишите программу, которая помещает в регистр J значение N, $0 < N \leq 3000$, при условии, что N содержится в rI4. Программа должна начинаться с адреса 3000. После выполнения программы содержимое всех ячеек памяти должно остаться неизменным.

▸ **22.** [*28*] В ячейке 2000 содержится целое число X. Напишите две программы, которые вычисляют X^{13} и останавливаются после занесения результата в регистр A. Одна программа должна занимать минимальное количество ячеек памяти MIX, а другая — выполняться за минимальное время. При этом предполагается, что X^{13} умещается в одном слове.

23. [*27*] В ячейке 0200 содержится слово

+	a	b	c	d	e

.

Напишите две программы, которые получают "отраженное" слово

+	e	d	c	b	a

и останавливаются после занесения этого результата в регистр A. В одной из программ не должна использоваться способность MIX загружать и сохранять частичные поля слов. Обе программы должны занимать минимальное количество ячеек памяти, которое возможно при данных условиях (включая ячейки, занимаемые самой программой, а также ячейки, используемые для временного хранения промежуточных результатов).

24. [*21*] Пусть в регистрах A и X содержатся

+	0	a	b	c	d

и

+	e	f	g	h	i

соответственно. Напишите две программы, которые меняют содержимое этих регистров на

+	a	b	c	d	e

и

+	0	f	g	h	i

соответственно и удовлетворяют двум условиям: (a) занимают минимальный объем памяти; (b) выполняются за минимальное время.

▸ **25.** [*30*] Предположим, фирма — производитель компьютера MIX планирует выпустить более мощный компьютер ("Mixmaster"?) и ей нужно убедить как можно больше владельцев компьютера MIX потратиться на более дорогой компьютер Причем фирма собирается разработать аппаратное обеспечение нового компьютера таким образом, чтобы он был *расширенным вариантом* MIX. Это означает, что все правильно написанные программы для MIX будут работать на новом компьютере и в них не нужно будет вносить каких-либо изменений. Внесите свои предложения по поводу того, что можно было бы воплотить в новом компьютере. (Например, *можете ли вы извлечь бóльшую пользу, применив поле* I команды?)

▸ **26.** [*32*] Эта задача состоит в написании программы загрузки перфокарт. На каждом компьютере процесс начальной загрузки, т. е. получения первоначальной информации и

корректного начала работы, выполняется по-разному. В случае использования компьютера MIX содержимое перфокарты можно считать только в символьном коде, и перфокарты, содержащие саму загружающую программу, тоже должны удовлетворять этому ограничению Поэтому с перфокарты можно считать не все возможные значения в байтах; кроме того, каждое слово, считываемое с перфокарт, является положительным.

У компьютера MIX есть одна функция, о которой не говорилось в тексте раздела. Речь идет о "кнопке GO", которая используется для начального запуска компьютера, когда в его памяти содержится произвольная информация. Когда оператор нажимает эту кнопку, выполняются следующие действия.

1) Одна карта считывается в ячейки 0000–0015; по сути, это эквивалентно команде "IN 0(16)".
2) Когда карта полностью прочитана и устройство чтения перфокарт больше не занято, осуществляется переход (JMP) в ячейку 0000. Кроме того, обнуляется содержимое регистра J.
3) Компьютер начинает выполнять программу, которую он считал с перфокарты.

Замечание. У компьютеров MIX, не имеющих устройств чтения перфокарт, кнопка GO находится на другом устройстве ввода Но в данной задаче предполагается наличие устройства чтения перфокарт под номером 16.

Программу загрузки нужно написать так, чтобы она удовлетворяла следующим условиям.

i) Вводимая колода должна начинаться с программы загрузки, за которой следуют перфокарты входных данных, содержащие загружаемые числа. Затем идет "переходная карта", которая закрывает программу загрузки и переходит к началу программы. Программа загрузки должна поместиться на двух перфокартах

ii) Перфокарты входных данных должны иметь следующий формат.

Колонки 1–5: программой загрузки игнорируются
Колонка 6: число последовательных слов, загружаемых с этой перфокарты (*лежит в промежутке от 1 до 7 включительно*).
Колонки 7–10: адрес ячейки, в которую загружается слово 1. Этот адрес всегда больше 100, чтобы не было перекрытия с программой загрузки.
Колонки 11–20: слово 1.
Колонки 21–30: слово 2 (если содержимое колонки 6 ≥ 2).
...
Колонки 71–80: слово 7 (если в колонке 6 содержится число 7).

Содержимое слов 1, 2, ... перфорируется и представляется в числовом виде по аналогии с десятичными числами. Если слово отрицательно, то знак "–" ("позиция 11") *перфорируется поверх* наименьшей значащей цифры, т. е в колонке 20. Предполагается, что по этой причине символьный код вводится в колонки 10, 11, 12, ..., 19, а не в 30, 31, 32, .. , 39. *Например, если на перфокарте в колонках 1–40 выбито*

$$\texttt{ABCDE3100001234567890000000001000000010}\overline{\texttt{0}},$$

то будут загружены следующие данные:

1000. +0123456789, 1001. +0000000001; 1002: −0000000100.

iii) На переходной карте в колонках 1–10 должна содержаться информация в формате TRANSOnnnn, где nnnn — адрес ячейки, с которой начинается выполнение программы.

iv) Программа загрузки должна работать независимо от размера байта, чтобы в содержащие ее перфокарты не нужно было вносить никаких изменений. На картах не должны содержаться символы, соответствующие байтам 20, 21, 48, 49, 50, ... (т. е. символы Σ,

П, =, $, <, ...), так как их не сможет прочитать ни одно устройство чтения перфокарт. В частности, нельзя использовать команды ENT, INC и CMP, поскольку их невозможно перфорировать на карте.

1.3.2. Язык ассемблера компьютера MIX

Символический язык компьютера MIX используется для того, чтобы облегчить чтение и написание программ, а также избавить программиста от беспокойства по поводу утомительных мелких деталей, которые часто становятся причиной дополнительных ошибок. Этот язык под названием MIXAL (MIX Assembly Language*) является расширенным вариантом системы обозначений, которая использовалась для команд в предыдущем разделе. Главной его особенностью является то, что для обозначения чисел можно использовать буквенные имена, а с помощью поля метки связывать имена с ячейками памяти.

Чтобы легче было разобраться в языке MIXAL, рассмотрим простой пример. Приведенный ниже код является частью большой программы; это подпрограмма нахождения максимума n элементов $X[1], \ldots, X[n]$ с помощью алгоритма 1.2.10M.

Программа M (*Нахождение максимума*). Занесение значений в регистры: rA $\equiv m$, rI1 $\equiv n$, rI2 $\equiv j$, rI3 $\equiv k$, $X[i] \equiv$ CONTENTS(X + i).

Машинный код						Номер строки	МЕТКА	ОП	АДРЕС	Повтор	Примечания
						01	X	EQU	1000		
						02		ORIG	3000		
3000:	+	3009	0	2	32	03	MAXIMUM	STJ	EXIT	1	Связь с подпрограммой.
3001:	+	0	1	2	51	04	INIT	ENT3	0,1	1	*M1. Инициализация.* $k \leftarrow n$.
3002:	+	3005	0	0	39	05		JMP	CHANGEM	1	$j \leftarrow n$, $m \leftarrow X[n]$, $k \leftarrow n-1$.
3003:	+	1000	3	5	56	06	LOOP	CMPA	X,3	$n-1$	*M3. Сравнение.*
3004:	+	3007	0	7	39	07		JGE	*+3	$n-1$	Перейти к шагу M5, если $m \geq X[k]$.
3005:	+	0	3	2	50	08	CHANGEM	ENT2	0,3	$A+1$	*M4. Изменение m.* $j \leftarrow k$.
3006:	+	1000	3	5	08	09		LDA	X,3	$A+1$	$m \leftarrow X[k]$.
3007:	+	1	0	1	51	10		DEC3	1	n	*M5. Уменьшение k.*
3008:	+	3003	0	2	43	11		J3P	LOOP	n	*M2. Все проверено?* Перейти к M3, если $k>0$.
3009:	+	3009	0	0	39	12	EXIT	JMP	*	1	Возврат к главной программе. ∎

Эта программа позволяет проиллюстрировать сразу несколько моментов.

a) Столбцы "МЕТКА", "ОП" и "АДРЕС" представляют особый интерес; в них содержится программа на символическом языке MIXAL, и ниже мы подробно рассмотрим эту программу.

b) В столбце "Машинный код" содержатся реальные цифровые команды машинного языка, соответствующие символическим командам языка MIXAL. MIXAL был разработан так, чтобы любую программу на этом языке можно было легко транслировать на цифровой машинный язык. Трансляция обычно выполняется другой компьютерной программой, которая называется *ассемблерной программой* или *ассемблером*. Таким образом, для программирования на машинном языке программист может использовать MIXAL, чтобы не определять эквивалентные цифровые коды команд вручную. В этой книге практически все программы для MIX написаны на языке MIXAL.

* Язык ассемблера компьютера MIX. — *Прим. перев.*

c) Столбец "Номер строки" является необязательным компонентом программы на языке MIXAL, в этой книге он включен в примеры программ просто для облегчения ссылок на различные части программы.

d) В столбце "Примечания" содержатся пояснения к программе и ссылки на шаги алгоритма 1 2.10М. Читателю следует сравнить этот алгоритм (с. 127) с приведенной выше программой. Обратите внимание, что при переводе алгоритма в код для MIX допущена некоторая "программистская вольность", например шаг М2 оказался последним. При "занесении значений в регистры" (см. начало программы М) показано, какие компоненты MIX соответствуют переменным, используемым в алгоритме.

e) Столбец "Повтор" будет полезен при изучении многих программ для MIX, которые рассматриваются в данной книге. Он представляет *профиль программы*, т. е. показывает, сколько раз команда из данной строки будет выполняться в ходе работы программы. Так, например, команда из строки 06 будет выполнена $n-1$ раз и т. д. На основании этой информации можно определить, сколько времени требуется на выполнение подпрограммы; оно равно $(5+5n+3A)u$, где A — величина, которая была тщательно проанализирована в разделе 1.2.10.

А теперь давайте обсудим запись программы М на языке MIXAL. Строка 01,

```
X EQU 1000,
```

говорит о том, что символ X будет *эквивалентом* числа 1 000. Эффект этого действия проявляется в строке 06, в которой цифровой эквивалент команды "CMPA X,3" имеет вид

+	1000	3	5	56

т. е. "CMPA 1000,3".

Строка 02 указывает, что следующие строки расположены, начиная с адреса 3000. Поэтому символ MAXIMUM, находящийся в поле METKA строки 03, становится эквивалентным числу 3000, символ INIT — числу 3001, символ LOOP — числу 3003 и т. д.

В строках с 03 по 12 поля ОП содержатся символические имена команд MIX: STJ, ENT3 и т. д. Но символические имена EQU и ORIG, которые находятся в столбце ОП строк 01 и 02, несколько отличаются от них; EQU и ORIG называются *псевдооперациями*, так как они являются операторами языка MIXAL, но не порождают машинных команд MIX. Псевдооперации предоставляют специальную информацию о символической программе и в то же время не являются командами самой машины MIX. Так, например, строка

```
X EQU 1000
```

только сообщает некоторые сведения *о* программе М; это не означает, что во время выполнения программы какой-либо переменной присваивается значение 1000. Обратите внимание, что для строк 01 и 02 машинные команды не порождаются.

Строка 03 — это команда "сохранить J", которая сохраняет содержимое регистра J в поле (0:2) ячейки EXIT. Другими словами, она сохраняет rJ в поле адреса команды, расположенной в строке 12.

Как уже упоминалось, программа М является частью большой программы. Если, например, в каком-нибудь месте этой программы встретится последовательность

команд

```
ENT1 100
JMP  MAXIMUM
STA  MAX
```

это приведет к вызову программы M со значением n, равным 100. В этом случае программа M найдет максимальный элемент среди $X[1], \ldots, X[100]$ и вернет управление команде "STA MAX", записав максимальное значение в rA, а его номер j — в rI2 (см. упр. 3).

Строка 05 передает управление строке 08. Строки 04, 05 и 06 в дополнительных объяснениях не нуждаются. В строке 07 вводится новое обозначение: звездочка (читается "текущий"), которая указывает на начальную ячейку текущей команды; поэтому запись "*+3" ("текущий плюс три") означает "три ячейки после начала текущей команды". Поскольку в строке 07 находится команда, начинающаяся с ячейки 3004, то "*+3" означает ссылку на ячейку 3007.

Оставшаяся часть символического кода не требует каких-либо разъяснений, так как говорит сама за себя. Обратите внимание, что в строке 12 снова появляется звездочка (см. упр. 2).

В следующем примере демонстрируются другие функции языка ассемблера. Задача заключается в том, чтобы вычислить и напечатать таблицу первых 500 простых чисел, расположив их в 10 столбцах по 50 чисел. На АЦПУ эта таблица будет напечатана следующим образом.

```
ПЕРВЫЕ ПЯТЬСОТ ПРОСТЫХ ЧИСЕЛ
      0002 0233 0547 0877 1229 1597 1993 2371 2749 3187
      0003 0239 0557 0881 1231 1601 1997 2377 2753 3191
      0005 0241 0563 0883 1237 1607 1999 2381 2767 3203
      0007 0251 0569 0887 1249 1609 2003 2383 2777 3209
      0011 0257 0571 0907 1259 1613 2011 2389 2789 3217
        :                                             :
      0229 0541 0863 1223 1583 1987 2357 2741 3181 3571
```

Воспользуемся следующим методом.

Алгоритм P (*Печать таблицы 500 простых чисел*). Этот алгоритм (рис. 14) состоит из двух частей: при выполнении шагов P1–P8 готовится внутренняя таблица 500 простых чисел, а при выполнении шагов P9–P11 результаты печатаются в таком виде, как показано выше. В последней части программы используются два "буфера", в которых создаются образы строк: пока печатается содержимое одного буфера, другой заполняется информацией.

P1. [Начать таблицу.] Присвоить PRIME[1] $\leftarrow 2$, N $\leftarrow 3$, J $\leftarrow 1$. (В этой программе N пробегает нечетные числа в поисках кандидатов в простые числа; J подсчитывает, сколько простых чисел уже найдено.)

P2. [N — простое число.] Присвоить J $\leftarrow$ J $+ 1$, PRIME[J] $\leftarrow$ N.

P3. [500 чисел найдены?] Если J $= 500$, перейти к шагу P9.

P4. [Увеличить N.] Присвоить N $\leftarrow$ N $+ 2$.

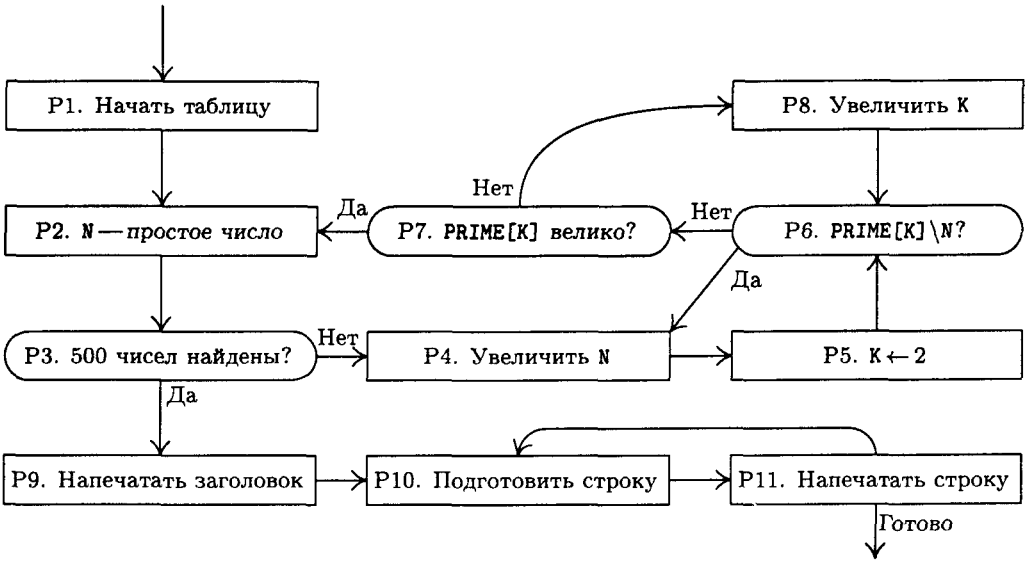


Рис. 14. Алгоритм P.

P5. [K ← 2.] Присвоить K ← 2. (PRIME[K] *пробегает возможные простые делители* N.)

P6. [PRIME[K]\N?] Разделить N на PRIME[K]; пусть Q — это частное от деления, а R — остаток. Если R = 0 (т. е. N не является простым), перейти к шагу P4.

P7. [PRIME[K] велико?] Если Q ≤ PRIME[K], перейти к шагу P2. (В таком случае N должно быть простым; доказательство этого факта интересное и немного необычное; см. упр. 6.)

P8. [Увеличить K.] Увеличить K на 1 и перейти к шагу P6.

P9. [Напечатать заголовок.] Теперь мы готовы к тому, чтобы напечатать таблицу. Переведем АЦПУ на следующую страницу. Занесем в BUFFER[0] строку заголовка и напечатаем эту строку. Присвоим B ← 1, M ← 1.

P10. [Подготовить строку.] Поместить PRIME[M], PRIME[50 + M], ..., PRIME[450 + M] в BUFFER[B] в соответствующем формате.

P11. [Напечатать строку.] Напечатать BUFFER[B]; присвоить B ← 1 − B (тем самым переключаясь на другой буфер) и увеличить M на 1. Если M ≤ 50, вернуться к шагу P10; в противном случае выполнение алгоритма заканчивается. ∎

Программа P (*Печать таблицы 500 простых чисел*). Данная программа написана несколько "топорным" способом, и это неспроста. Причина в том, что преследовалась цель — проиллюстрировать в одной программе большинство возможностей MIXAL. rI1 ≡ J − 500; rI2 ≡ N; rI3 ≡ K; rI4 указывает на B; rI5 равно M плюс число, кратное 50.

```
01 * ПРИМЕР ПРОГРАММЫ  ...  ТАБЛИЦА ПРОСТЫХ ЧИСЕЛ
02 *
03 L        EQU  500          Искомое количество простых чисел.
04 PRINTER  EQU  18           Номер АЦПУ.
05 PRIME    EQU  -1           Память для таблицы простых чисел.
06 BUF0     EQU  2000         Память для BUFFER[0].
07 BUF1     EQU  BUF0+25      Память для BUFFER[1].
08          ORIG 3000
09 START    IOC  0(PRINTER)   Перейти к новой странице.
10          LD1  =1-L=        P1. Начать таблицу. J ← 1.
11          LD2  =3=              N ← 3
12 2H       INC1 1            P2. N — простое число. J ← J + 1.
13          ST2  PRIME+L,1        PRIME[J] ← N.
14          J1Z  2F           P3. 500 чисел найдены?
15 4H       INC2 2            P4. Увеличить N.
16          ENT3 2            P5. K ← 2.
17 6H       ENTA 0            P6. PRIME[K]\N?
18          ENTX 0,2              rAX ← N.
19          DIV  PRIME,3          rA ← Q,  rX ← R.
20          JXZ  4B               Перейти к P4, если R = 0.
21          CMPA PRIME,3      P7. PRIME[K] велико?
22          INC3 1            P8. Увеличить K.
23          JG   6B               Перейти к P6, если Q > PRIME[K].
24          JMP  2B               В противном случае N — простое.
25 2H       OUT  TITLE(PRINTER) P9. Напечатать заголовок.
26          ENT4 BUF1+10          Присвоить B ← 1.
27          ENT5 -50              Присвоить M ← 0.
28 2H       INC5 L+1              Увеличить M
29 4H       LDA  PRIME,5      P10. Подготовить строку
30          CHAR                  Преобразовать PRIME[M] в
31          STX  0,4(1:4)             десятичный формат.
32          DEC4 1
33          DEC5 50               (rI5 уменьшается на 50, пока
34          J5P  4B                   не станет неположительным.)
35          OUT  Q,4(PRINTER) P11. Напечатать строку.
36          LD4  24,4             Переключить буфера.
37          J5N  2B               Если rI5 = 0, то работа алгоритма
38          HLT                       заканчивается.
39 * ПЕРВОНАЧАЛЬНОЕ СОДЕРЖИМОЕ ТАБЛИЦ И БУФЕРОВ
40          ORIG PRIME+1
41          CON  2            Первое простое число — 2.
42          ORIG BUF0-5
43 TITLE    ALF  FIRST        Символы для
44          ALF   FIVE            строки заголовка.
45          ALF   HUND
46          ALF  RED P
47          ALF  RIMES
48          ORIG BUF0+24
49          CON  BUF1+10      Буфера ссылаются один на другой.
50          ORIG BUF1+24
```

```
51          CON  BUF0+10
52          END  START          Конец программы.
```

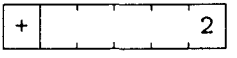
В отношении этой программы необходимо отметить следующие интересные моменты.

1. Строки 01, 02 и 39 начинаются со звездочки. Таким образом обозначается строка комментария, которая содержит только пояснения и не оказывает реального воздействия на транслируемую программу.

2. Как и в программе M, псевдооперация EQU из строки 03 определяет эквивалент символа. В данном случае эквивалентом L назначено число 500. (В строках 10–24 этой программы L представляет количество простых чисел, которые нужно найти.) Обратите внимание, что в строке 05 символу PRIME присваивается *отрицательный* эквивалент; вообще говоря, эквивалентом символа может быть любое число, состоящее из пяти байтов и знака. В строке 07 эквивалент BUF1 вычисляется по формуле BUF0+25, что в результате дает 2025. В MIXAL арифметические операции над числами можно выполнять в ограниченном объеме. Еще один пример арифметической операции появляется в строке 13, в которой ассемблер вычисляет значение PRIME+L (в данном случае это 499).

3. Символ PRINTER в строках 25 и 35 используется в F-части. F-часть, которая всегда заключается в круглые скобки, может состоять из чисел либо символов точно так, как другие части поля ADDRESS. В строке 31 иллюстрируется спецификация частичного поля "(1:4)", в котором используется двоеточие.

4. В MIXAL предусмотрено несколько способов определения слов, которые не являются командами. В строке 41 псевдооперация CON используется для определения обычной константы, "2". В результате трансляции строки 41 получится слово



.

В строке 49 присутствует немного более сложная константа, "BUF1+10", в результате трансляции которой получится слово

.

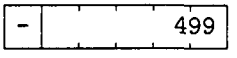
Константа может быть заключена между знаками равенства (см. строки 10 и 11); в этом случае она называется *литералом (или буквенной константой)*. Ассемблер автоматически создает для буквенных констант внутренние имена и вставляет строки "CON". Например, в результате трансляции строк 10 и 11 программы P получится

```
10          LD1  con1
11          LD2  con2
```

а в конце программы, между строками 51 и 52, в результате трансляции будут вставлены строки

```
51a  con1  CON  1-L
51b  con2  CON  3
```

Трансляция строки 51а дает слово



.

Использовать буквенные константы (литералы), несомненно, удобно, так как программистам не нужно изобретать для тривиальных констант символические имена, а также помнить о том, что в конце каждой программы необходимо вставлять константы. Таким образом, программист может сосредоточиться на главной задаче и не волноваться по поводу подобных деталей. (Однако нужно заметить, что примеры литералов в программе P не слишком удачны, поскольку, заменив строки 10 и 11 более эффективными командами "ENT1 1-L" и "ENT2 3", мы несколько улучшили бы программу.)

5. Хороший язык ассемблера должен имитировать ход *мыслей* программиста при написании машинных программ. Одним из примеров этой философии является использование литералов, о которых только что шла речь. В качестве другого примера можно привести применение символа "*", которое обсуждалось при описании программы M. А третьим примером является идея использования *локальных символов*, таких как символ 2H, который появляется в поле метки строк 12, 25 и 28.

Локальные символы — это специальные символы, которые можно *переопределять* столько раз, сколько нужно. Глобальный символ, например PRIME, имеет только одно значение на протяжении всей программы, и если бы он появился в поле метки более чем одной строки, то ассемблер зафиксировал бы ошибку. Но локальные символы имеют различную природу; например, мы пишем 2H ("2 here" — "2 здесь") в поле метки и 2F ("2 forward" — "2 вперед") или 2B ("2 backward" — "2 назад") в адресном поле строки MIXAL:

2B означает ближайшую *предыдущую* метку 2H;
2F означает ближайшую *следующую* метку 2H.

Таким образом, символ "2F" в строке 14 означает ссылку на строку 25, символ "2B" в строке 24 — ссылку назад, на строку 12, а символ "2B" в строке 37 — ссылку на строку 28. Адрес 2F или 2B никогда не относится к *собственной* строке. Например, три строки кода MIXAL

```
2H    EQU   10
2H    MOVE  2F(2B),
2H    EQU   2B-3
```

в сущности, эквивалентны одной строке

```
MOVE *-3(10).
```

Символы 2F и 2B никогда не следует использовать в поле метки, а символ 2H — в поле адреса. Существует десять локальных символов, которые можно получить, заменив в этих примерах "2" любой другой цифрой от 0 до 9.

Идею локальных символов выдвинул М. Э. Конвей (M. E. Conway) в 1958 году в связи с разработкой ассемблера для машины UNIVAC I. Локальные символы освобождают программиста от необходимости выбора символических имен для каждого адреса, когда нужно всего лишь сослаться на команду, находящуюся на расстоянии нескольких строк. Соседним командам не всегда можно придумать подходящие имена, поэтому программисты склонны вводить такие лишенные содержания символы,

как X1, X2, X3 и т. д., что создает потенциальную опасность их повторения. Поэтому использовать локальные символы в языке ассемблера очень полезно и совершенно естественно.

6. Адресная часть строк 30 и 38 пуста. Это означает, что в результате трансляции получится нулевой адрес. В строке 17 адресную часть тоже можно было бы оставить пустой, но без этого лишнего 0 программа стала бы менее наглядной.

7. В строках 43–47 используется операция ALF, которая создает пятибайтовую константу в буквенно-цифровом символьном коде MIX. Например, в результате трансляции строки 45 получится слово

+	00	08	24	15	04

т. е. "␣HUND" (часть слова "пятьсот") — часть строки заголовка в выходных данных программы P.

Все ячейки, содержимое которых не определено в программе MIXAL, обычно обнуляются (за исключением ячеек, которые используются загружающей программой; обычно это ячейки 3700–3999). Поэтому после строки 47 нет необходимости определять пробелы в других словах заголовка.

8. Вместе с операцией ORIG можно выполнять арифметические действия (см. строки 40, 42 и 48).

9. В последней строке законченной программы на языке MIXAL всегда присутствует код операции END. Адрес в этой строке указывает на ячейку, с которой начинает выполняться программа после загрузки в память.

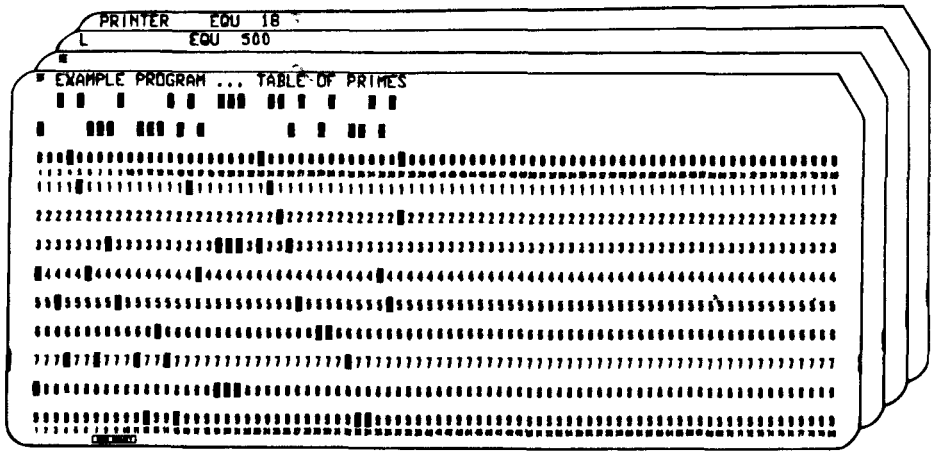
10. И в завершение анализа программы P следует отметить, что ее команды построены так, чтобы значения в индексных регистрах можно было сравнивать с нулем, когда это только возможно. Например, в регистре rI1 сохраняется величина J-500, а не J. Особого внимания в этом смысле заслуживают строки 26–34, хотя разобраться в них, наверное, непросто.

Интересно отметить статистические характеристики, которые наблюдаются во время реальной работы программы P. Команда деления из строки 19 была выполнена 9 538 раз, а время выполнения строк 10–24 составило $182144u$.

Программы на языке MIXAL можно перфорировать на картах или набрать на терминале компьютера, как показано на рис. 15. При использовании перфокарт выбирается следующий формат.

Колонки 1–10	Поле МЕТКА
Колонки 12–15	Поле ОП
Колонки 17–80	Поле АДРЕС и необязательные примечания
Колонки 11, 16	Пустые

Но если в колонке 1 содержится звездочка, то все содержимое перфокарты рассматривается как комментарий. Поле АДРЕС заканчивается первой же пустой колонкой (пробелом), следующей за колонкой 16. Справа от этой первой колонки можно




```
* ПРИМЕР ПРОГРАММЫ...  ТАБЛИЦА ПРОСТЫХ ЧИСЕЛ
*
L EQU 500
PRINTER EQU 18
PRIME EQU -1
BUF0 EQU 2000
BUF1 EQU BUF0+25
 ORIG 3000
START IOC 0(PRINTER)
 LD1 =1-L=
```

Рис. 15. Первые строки программы P, перфорированные на картах либо набранные на терминале.

перфорировать любые комментарии, которые не влияют на транслируемую программу. (*Исключение.* Если в поле ОП содержится команда ALF, то примечания всегда начинаются в колонке 22.)

Когда входные данные вводятся с терминала, используется менее ограничительный формат: поле МЕТКА заканчивается первым же пробелом, а поля ОП и АДРЕС (если они есть) начинаются непустым символом и продолжаются до следующего пробела. В то же время за особым кодом операции ALF следуют либо два пробела и пять буквенно-цифровых символов, либо один пробел и пять буквенно-цифровых символов, первый из которых не является пробелом. В оставшейся части каждой строки могут содержаться примечания.

Ассемблер MIX берет подготовленные таким образом файлы и преобразует их в загрузочные модули программ на машинном языке. При благоприятном стечении обстоятельств читатель сможет получить доступ к ассемблеру MIX и имитатору MIX и проработать различные упражнения из этой книги.

Теперь вы знаете, что можно сделать с помощью языка MIXAL. В заключение этого раздела мы дадим более подробное описание правил и, в частности, обратим внимание на то, чего *не* разрешается делать на языке MIXAL. Фактически язык определяется сравнительно небольшим количеством правил, приведенных ниже.

1. *Символ* — это строка, содержащая от одной до десяти букв и/или цифр, среди которых должна быть по крайней мере одна буква. *Примеры* PRIME, TEMP, 20BY20. Специальные символы dH, dF и dB, где d — это одна цифра, в целях данного определения будут заменяться другими уникальными символами в соответствии с соглашением о "локальных символах", о котором говорилось выше.

2. *Число* — это строка, содержащая от одной до десяти цифр. *Пример:* 00052.

3. В каждом случае появления в программе MIXAL символ называется либо определенным символом, либо ссылкой вперед. *Определенный символ* — это символ, который появляется в поле METKA одной из предыдущих строк программы MIXAL. *Ссылка вперед* — это символ, который пока еще не был определен подобным образом.

4. *Элементарное выражение* — это либо

a) число, либо

b) определенный символ (обозначающий числовой эквивалент этого символа, см. правило 13), либо

c) звездочка (обозначающая значение ⊛; см. правила 10 и 11).

5. *Выражение* — это либо

a) элементарное выражение, либо

b) элементарное выражение, перед которым стоит знак "+" или "−", либо

c) выражение, за которым следует бинарная операция, а за ней — элементарное выражение.

Допустимыми являются шесть следующих бинарных операций: +, -, *, /, // и :. Они определяются для числовых слов MIX следующим образом.

```
C = A+B     LDA AA; ADD BB; STA CC.
C = A-B     LDA AA; SUB BB; STA CC.
C = A*B     LDA AA; MUL BB; STX CC.
C = A/B     LDA AA; SRAX 5; DIV BB; STA CC.
C = A//B    LDA AA; ENTX 0; DIV BB; STA CC.
C = A:B     LDA AA; MUL =8=; SLAX 5; ADD BB; STA CC.
```

Здесь AA, BB и CC — ячейки, содержащие соответствующие значения символов A, B и C. Операции внутри выражения выполняются слева направо. *Примеры:*

-1+5	равно 4.
-1+5*20/6	равно 4*20/6 равно 80/6 равно 13 (операции выполняются слева направо).
1//3	равно слову MIX, размер которого приблизительно равен $b^5/3$, где b — размер байта; т. е. слово, представляющее дробь $\frac{1}{3}$ с десятичной точкой слева.
1:3	равно 11 (обычно используется в частичной спецификации поля).
*-3	равно ⊛ минус три.
***	равно ⊛, умноженному на ⊛.

6. *A-часть* (которая используется для описания адресного поля команды MIX) либо

a) пуста (и обозначает нулевое значение), либо

b) является выражением, либо

c) является ссылкой вперед (и обозначает окончательный эквивалент символа; *см. правило* 13), *либо*

d) является литералом (и обозначает ссылку на внутренний символ; см. правило 12).

7. *Индексная часть* (которая используется для описания индексного поля команды MIX), либо

a) пуста (и обозначает нулевое значение), либо

b) состоит из запятой и следующего за ней выражения (и обозначает значение этого выражения).

8. *F-часть* (которая используется для описания F-поля команды MIX) либо

a) пуста (что обозначает стандартное F-значение в зависимости от содержимого поля ОП (см. табл. 1.3.1–1)), либо

b) состоит из выражения, заключенного в круглые скобки (и обозначает значение этого выражения).

9. *W-значение* (которое используется для описания константы MIX, занимающей *полное слово*) — это *либо*

a) выражение, за которым следует F-часть (в этом случае пустая F-часть обозначается через (0:5)), либо

b) W-значение, за которым после запятой следует W-значение вида (a).

W-значение указывает числовое значение слова MIX, которое определяется следующим образом. Пусть W-значение имеет вид "$E_1(F_1),E_2(F_2),\ldots,E_n(F_n)$", где $n \geq 1$, E_i — выражения, а F_i — поля. Желаемый результат — окончательное значение, которое появилось бы в ячейке памяти WVAL после выполнения следующей гипотетической программы:

$$\texttt{STZ WVAL; LDA } C_1\texttt{; STA WVAL(}F_1\texttt{); ...; LDA } C_n\texttt{; STA WVAL(}F_n\texttt{).}$$

Здесь $C_1, \ldots, C_n$ обозначают ячейки, содержащие значения выражений $E_1, \ldots, E_n$. Каждое F_i должно иметь вид $8L_i + R_i$, где $0 \leq L_i \leq R_i \leq 5$. *Примеры:*

1	слово	+||1
1,-1000(0:2)	слово	-|1000|1
-1000(0:2),1	слово	+||1

10. В процессе трансляции используется величина, которая обозначается через ✱ (и называется *счетчиком адреса*). Первоначальное значение счетчика адреса равно нулю. Значение ✱ всегда должно быть неотрицательным числом, которое помещается в двух байтах. Если в строке поле метки не пусто, то оно должно содержать символ, который не был определен ранее. Эквивалент этого символа затем определяется как текущее значение ✱.

11. После обработки поля METKA, как описано в правиле 10, процесс трансляции будет зависеть от значения содержимого поля ОП. Существует шесть возможностей для ОП.

a) В поле ОП содержится символический оператор MIX. В табл. 1 из предыдущего раздела определены стандартные значения C и F для каждого оператора MIX. В этом случае в поле АДРЕС должна находиться A-часть (правило 6), за которой следует индексная часть (правило 7), а затем — F-часть (правило 8). Таким образом, получаем четыре значения: C, F, A и I. В результате транслируется слово, которое определяется последовательностью "LDA C; STA WORD; LDA F; STA WORD(4:4); LDA I; STA WORD(3:3); LDA A; STA WORD(0:2)" и помещается в ячейку, заданную ⊛, а затем увеличивается на 1 значение счетчика ⊛.

b) В поле ОП содержится операция "EQU". В поле АДРЕС должно содержаться W-значение (см. правило 9). Если поле METKA не пусто, то значение содержащегося здесь символа устанавливается равным значению, заданному в поле АДРЕС. Это правило имеет более высокий приоритет, чем правило 10. Значение ⊛ не меняется. (В качестве нетривиального примера рассмотрим строку

```
BYTESIZE  EQU  1(4:4),
```

позволяющую программисту получить символ, значение которого зависит от размера байта. Эта ситуация допустима до тех пор, пока программа имеет смысл для всех возможных размеров байта.)

c) В поле ОП находится "ORIG". В поле АДРЕС должно содержаться W-значение (см. правило 9); значение счетчика адреса ⊛ устанавливается равным этому значению. (Заметьте, что согласно правилу 10 символ, находящийся в поле METKA строки с операцией ORIG, принимает значение ⊛ *до* его изменения. Например,

```
TABLE    ORIG  *+100
```

делает символ TABLE эквивалентным текущему адресу плюс 100.)

d) В поле ОП находится "CON". В поле АДРЕС должно содержаться W-значение. В результате происходит трансляция слова, имеющего это значение, помещение его в ячейку, заданную ⊛, и увеличение значения счетчика ⊛ на 1.

e) В поле ОП находится "ALF". В результате выполняется трансляция слова из символьных кодов, образуемого первыми пятью символами адресного поля; в остальном операция аналогична CON.

f) В поле ОП находится "END". В поле ADDRESS должно содержаться W-значение, определяющее в своем поле (4:5) адрес команды, с которой начинается программа. Строка END обозначает окончание программы на языке MIXAL. В завершение ассемблер вставляет в произвольном порядке непосредственно перед строкой END дополнительные строки, соответствующие всем неопределенным символам и литеральным константам (см. правила 12 и 13). Таким образом, символ в поле METKA строки END будет обозначать первую ячейку, следующую за вставленными словами.

12. Литеральные константы. W-значение, длина которого — менее 10 символов, можно заключить между знаками "=" и использовать в качестве ссылки вперед.

В результате будет создан новый внутренний символ и сразу перед строкой END будет вставлена строка CON, определяющая этот символ (см. примечание 4 после программы P).

13. Каждому символу соответствует одно и только одно значение. Это число, занимающее полное слово MIX, обычно определяется символом из поля METKA в соответствии с правилом 10 или 11, (b). Если этого символа не было в поле METKA, то перед строкой END вставляется новая строка, у которой ОП = "CON", АДРЕС = "0" и в поле LOC которой содержится имя символа.

Замечание. Самым важным следствием из приведенных выше правил является ограничение на ссылки вперед. Для этого нельзя использовать символ, который еще не был определен в поле LOC одной из предыдущих строк; его можно применять только в качестве A-части команды. В частности, этот символ нельзя использовать (a) в связи с арифметическими операциями или (b) в поле АДРЕС операций EQU, ORIG и CON. Например, операции

```
LDA  2F+1
```

и

```
CON  3F
```

недопустимы. Это ограничение было наложено для того, чтобы обеспечить более эффективную трансляцию программ. Кроме того, опыт, полученный в процессе написания данной серии книг, показал, что это очень мягкое ограничение, которое редко имеет сколько-нибудь существенное значение.

На самом деле у MIX есть два символических языка программирования низкого уровня: MIXAL*, машинно-ориентированный язык, предназначенный для облегчения трансляции за один проход с помощью очень простого ассемблера, и PL/MIX, который более адекватно отражает информационные и управляющие структуры и выглядит, как поле примечаний программ на языке MIXAL. PL/MIX будет описан в главе 10.

УПРАЖНЕНИЯ (часть 1)

1. [*00*] В тексте раздела отмечалось, что запись "X EQU 1000" не генерирует машинной команды, которая присваивает значение переменной. Предположим, вы пишете программу для MIX, в которой хотите присвоить значение, равное 1 000, некоторой ячейке памяти (с символическим именем X). Как это сделать на языке MIXAL?

▸ **2.** [*10*] Строка 12 программы M выглядит так: "JMP *", где * — метка этой же строки. Почему программа не зацикливается, вновь и вновь повторяя эту команду?

▸ **3.** [*23*] Каким будет результат выполнения следующей программы, если она используется вместе с программой M?

```
START IN   X+1(0)
      JBUS *(0)
      ENT1 100
1H    JMP  MAXIMUM
      LDX  X,1
```

* Автор был крайне удивлен, когда в 1971 году узнал о том, что MIXAL — это еще и название югославского стирального порошка для *автоматических* стиральных машин.

```
        STA  X,1
        STX  X,2
        DEC1 1
        J1P  1B
        OUT  X+1(1)
        HLT
        END  START  ▌
```

▶ **4.** [*25*] Выполните трансляцию программы P вручную. (Это займет не так много времени, как вы думаете.) Какие числа будут фактически содержаться в оперативной памяти в соответствии с этой символической программой?

5. [*11*] Почему в программе P не нужна команда JBUS для определения времени готовности АЦПУ?

6. [*HM20*] (a) Покажите, что если n не является простым числом, то n имеет делитель d, такой, что $1 < d \leq \sqrt{n}$. (b) Учитывая этот факт, докажите, что проверка на шаге P7 алгоритма P показывает, что N — простое число.

7. [*10*] (a) Что означает "4B" в строке 34 программы P? (b) Каким будет эффект (если он вообще будет), если метку строки 15 заменить меткой "2H", а адрес строки 20 — адресом "2B"?

▶ **8.** [*24*] Что делает следующая программа? (Не запускайте ее на компьютере, найдите ответ "вручную"!)

```
        * ЗАГАДОЧНАЯ ПРОГРАММА
BUF     ORIG *+3000
1H      ENT1 1
        ENT2 0
        LDX  4F
2H      ENT3 0,1
3H      STZ  BUF,2
        INC2 1
        DEC3 1
        J3P  3B
        STX  BUF,2
        INC2 1
        INC1 1
        CMP1 =75=
        JL   2B
        ENN2 2400
        OUT  BUF+2400,2(18)
        INC2 24
        J2N  *-2
        HLT
4H      ALF  AAAAA
        END  1B  ▌
```

УПРАЖНЕНИЯ (часть 2)

Эти упражнения представляют собой небольшие задачи по программированию, иллюстрирующие типичные примеры применения компьютеров и охватывающие широкий диапазон различных методов. Читателю настоятельно рекомендуется решить хотя бы несколько из этих задач, чтобы получить некоторый опыт использования MIX, а также

достаточно полное представление об основах искусства программирования. Если хотите, можете проработать эти упражнения одновременно с чтением оставшейся части главы 1.

Ниже перечислены используемые в задачах методы программирования.

Использование таблиц-переключателей для многовариантных решений: упр. 9, 13 и 23.

Использование индексных регистров и двумерных массивов: упр. 10, 21 и 23.

Распаковка символов: упр. 13 и 23.

Целочисленная и десятичная арифметика и масштабирование: упр. 14, 16 и 18.

Использование подпрограмм: упр. 14 и 20.

Буферизация ввода: упр. 13.

Буферизация вывода: упр. 21 и 23.

Обработка списков: упр. 22.

Управление в реальном времени: упр. 20.

Графическое отображение: упр. 23.

Каждый раз, когда в упражнении из этой книги говорится "напишите программу для MIX" или "напишите подпрограмму для MIX", следует написать только символический код на языке MIXAL для решения требуемой задачи. Этот код будет представлять собой не законченную программу, а только фрагмент полной (гипотетической) программы. В фрагменте кода не нужно выполнять ввод или вывод данных, если они должны быть взяты из внешних источников. Необходимо использовать только следующие поля строк программы на языке MIXAL: МЕТКА, ОП и АДРЕС, а также снабдить их соответствующими комментариями. Числовое представление транслированных машинных команд, номера строк и колонки с итоговым количеством выполнений различных команд (см. программу M) указывать не нужно, за исключением случаев, когда это оговаривается особо. Строка END тоже не нужна.

С другой стороны, если в упражнении говорится "напишите *полную* программу для MIX", значит, нужно написать выполняемую программу на языке MIXAL, которая, в частности, должна включать финальную строку END. Ассемблеры и имитаторы MIX, на которых можно протестировать полные программы, широко распространены, поэтому проблем возникнуть не должно.

▸ **9.** [*25*] В ячейке INST содержится слово MIX, которое предположительно является командой MIX. Напишите программу для MIX, в которой совершается переход к ячейке GOOD, если это слово имеет допустимое C-поле, допустимое ±AA-поле, допустимое I-поле и допустимое F-поле в соответствии с табл. 1.3.1–1. В противном случае в программе должен быть выполнен переход к ячейке BAD. Помните, что допустимость F-поля зависит от C-поля — например, если C = 7 (MOVE), то допустимо любое F-поле, но если C = 8 (LDA), то F-поле должно иметь вид 8L + R, где $0 \le L \le R \le 5$. ±AA-поле считается допустимым, за *исключением* случая, когда C определяет команду, запрашивающую адрес памяти, I = 0 и ±AA не является допустимым адресом памяти.

Замечание. Неопытные программисты склонны решать подобную задачу, программируя длинные серии тестов C-поля, такие как "LDA C; JAZ 1F; DECA 5; JAN 2F; JAZ 3F; DECA 2; JAN 4F; ...". Этот подход никуда *не* годится! Лучший способ выбрать один из многих вариантов — подготовить вспомогательную *таблицу*, содержащую информацию, в которой сосредоточена необходимая логическая схема. Если бы эта таблица содержала, например, 64 элемента, мы бы написали "LD1 C; LD1 TABLE,1; JMP 0,1" и в результате очень быстро перешли бы к нужной программе. В подобной таблице можно хранить и другую полезную информацию. Табличный подход к решению данной задачи лишь ненамного увеличивает длину программы (из-за включения таблицы), но зато существенно повышает скорость ее выполнения и делает более универсальной.

▸ **10.** [*31*] Пусть имеется матрица размера 9×8

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{18} \\ a_{21} & a_{22} & a_{23} & \dots & a_{28} \\ \vdots & & & & \vdots \\ a_{91} & a_{92} & a_{93} & \dots & a_{98} \end{pmatrix},$$

которая хранится в памяти так, что a_{ij} находится в ячейке $1000 + 8i + j$. Поэтому ячейки памяти, в которых хранится эта матрица, выглядят следующим образом:

$$\begin{pmatrix} (1009) & (1010) & (1011) & \dots & (1016) \\ (1017) & (1018) & (1019) & \dots & (1024) \\ \vdots & & & & \vdots \\ (1073) & (1074) & (1075) & \dots & (1080) \end{pmatrix}$$

Говорят, что матрица имеет "седловую точку", если некоторый элемент является минимальным значением в строке и максимальным — в столбце. Математически это можно выразить так: элемент a_{ij} является седловой точкой данной матрицы, если

$$a_{ij} = \min_{1 \le k \le 8} a_{ik} = \max_{1 \le k \le 9} a_{kj}.$$

Напишите программу для MIX, которая определяет адрес седловой точки (если в матрице есть по крайней мере одна такая точка) или выдает нулевое значение (если седловой точки нет) и заканчивает работу, поместив найденный результат в rI1.

11. [*M29*] Чему равна *вероятность* того, что матрица из предыдущего упражнения имеет седловую точку, если ее 72 элемента различны и все $72!$ варианта одинаково возможны? Чему будет равна соответствующая вероятность, если матрица состоит только из нулей и единиц и все 2^{72} вариантов таких матриц являются одинаково возможными?

12. [*HM42*] К упр. 10 даны два решения (см. с. 570) и предложено найти еще одно, но не ясно какое из них лучше. Проанализируйте эти алгоритмы на основании предположений из упр 11 и решите, какой метод лучше.

13. [*28*] Дешифровщику необходимо подсчитать частоту появления букв в некотором закодированном тексте. Этот текст перфорирован на бумажной ленте; о его окончании сигнализирует символ "звездочка" Напишите полную программу для MIX, которая считывает данные с перфоленты, подсчитывает частоту появления каждого символа вплоть до первой звездочки, а затем печатает результаты в виде

```
A    0010257
B    0000179
D    0794301
```

и т д . по одному символу в строке Не нужно подсчитывать количество пробелов, а также печатать результаты для символов, частота появления которых равна нулю (например, для таких, как символ C в нашем примере) В целях эффективности используйте "буферизацию" при вводе во время считывания блока в одну область памяти можно выполнять подсчет символов в другой области Будем предполагать, что на ленте, содержащей входные данные, есть еще один дополнительный блок (следующий за тем, в котором находится заключительная звездочка).

▸ **14.** [*31*] Следующий алгоритм, который был получен неаполитанским астрономом Алоизием Лилиусом (Aloysius Lilius) и немецким математиком иезуитом Кристофером Клавиусом (Christopher Clavius) в конце 16 века, используется большинством западных христианских церквей для определения даты пасхального воскресенья для любого года после 1582.

Алгоритм E (*Определение даты пасхи*). Пусть Y — это год, для которого нужно получить дату пасхи.

E1. [Золотое число.] Присвоить $G \leftarrow (Y \bmod 19)+1$. ($G$ — это так называемое "золотое число" года в 19-летнем лунно-солнечном цикле.)

E2. [Столетие.] Присвоить $C \leftarrow \lfloor Y/100 \rfloor + 1$. (Если Y не кратно 100, то C — номер столетия, например 1984 год принадлежит двадцатому столетию.)

E3. [Поправки.] Присвоить $X \leftarrow \lfloor 3C/4 \rfloor - 12$, $Z \leftarrow \lfloor (8C+5)/25 \rfloor - 5$. (Здесь X — число годов, таких как 1900, когда дополнительный день високосного года не добавляется, чтобы "идти в ногу" с Солнцем; Z — специальная поправка, предназначенная для синхронизации даты пасхи с орбитой Луны.)

E4. [Поиск воскресенья.] Присвоить $D \leftarrow \lfloor 5Y/4 \rfloor - X - 10$. [$((-D) \bmod 7)$ марта действительно будет воскресеньем.]

E5. [Эпакта.] Присвоить $E \leftarrow (11G + 20 + Z - X) \bmod 30$. Если $E = 25$ и золотое число G больше 11 или если $E = 24$, то увеличить E на 1. (Это число E — *эпакта*, которая определяет дату полнолуния.)

E6. [Поиск полнолуния.] Присвоить $N \leftarrow 44 - E$. Если $N < 21$, то присвоить $N \leftarrow N + 30$. (Считается, что пасха — это первое воскресенье, следующее за первым полнолунием, которое произошло не ранее 21 марта. На самом деле это определение не является абсолютно точным из-за возмущений лунной орбиты, но в данном случае нас интересует не реальная, а "календарная" луна. N-е марта — это календарное полнолуние.)

E7. [Перейти к воскресенью.] Присвоить $N \leftarrow N + 7 - ((D + N) \bmod 7)$.

E8. [Определить месяц.] Если $N > 31$, то искомой датой будет $(N - 31)$ `АПРЕЛЯ`; в противном случае — N `МАРТА`. ▮

Напишите подпрограмму вычисления и печати даты пасхи для заданного года, предполагая, что номер этого года не превышает 100000. Выходные данные должны иметь вид "*dd* `МЕСЯЦ`, *yyyyy*", где *dd* — день, а *yyyyy* — год. Напишите полную программу для `MIX`, в которой указанная подпрограмма используется для построения таблицы дат пасхи с 1950 по 2000 год.

15. [*M30*] Достаточно распространенная ошибка при решении предыдущего упражнения состоит в непонимании того, что величина $(11G + 20 + Z - X)$ на шаге E5 может быть отрицательной; в результате положительный остаток mod 30 может быть вычислен неправильно. (См. *CACM* **5** (1962), 556.) Например, для года 14250 мы нашли бы $G = 1$, $X = 95$, $Z = 40$, поэтому, если бы у нас было $E = -24$ вместо $E = +6$, мы получили бы нелепый результат: "`42 АПРЕЛЯ`". Напишите полную программу для `MIX`, определяющую *самый ранний* год, для которого эта ошибка стала бы причиной неправильного вычисления даты пасхи.

16. [*31*] В разделе 1.2.7 было показано, что ряд $1 + \frac{1}{2} + \frac{1}{3} + \cdots$ расходится. Но если эту сумму вычислить на компьютере с ограниченной точностью, то в некотором смысле можно получить сходящуюся сумму, так как в конце концов ее члены окажутся настолько малыми, что уже ничего не будут добавлять к сумме. Для примера вычислим сумму с точностью до одного десятичного знака; получим $1 + 0.5 + 0.3 + 0.3 + 0.2 + 0.2 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 = 3.9$.

А если говорить более строго, пусть $r_n(x)$ — это число x, округленное с точностью до n десятичных знаков; тогда $r_n(x) = \lfloor 10^n x + \frac{1}{2} \rfloor / 10^n$ Теперь нужно найти

$$S_n = r_n(1) + r_n(\tfrac{1}{2}) + r_n(\tfrac{1}{3}) + \cdots.$$

Известно, что $S_1 = 3.9$ и задача состоит в том, чтобы написать полную программу для MIX, которая вычисляет и печатает значения S_n для $n = 2$, 3, 4 и 5

Замечание. Для этого существует гораздо более быстрый способ, чем простая процедура добавления членов $r_n(1/m)$ по одному, пока $r_n(1/m)$ не обратится в нуль Например, для всех значений m от 66667 до 200000 имеем $r_5(1/m) = 0.00001$, и значит, все 133334 раза можно избежать вычисления $1/m$! Необходимо использовать алгоритм, содержащий следующие строки.

A Начать с $m_h = 1$, $S = 1$.

B Присвоить $m_e = m_h + 1$ и вычислить $r_n(1/m_e) = r$.

C. Найти m_h — наибольшее m, для которого $r_n(1/m) = r$

D. Добавить $(m_h - m_e + 1)r$ к S и вернуться к шагу B.

17. [*HM30*] Используя обозначения из предыдущего упражнения, докажите или опровергните формулу

$$\lim_{n\to\infty}(S_{n+1} - S_n) = \ln 10$$

18. [*25*] Возрастающая последовательность всех несократимых дробей между 0 и 1, знаменатели которых не превосходят n, называется рядом Фарея порядка n Например, рядом Фарея порядка 7 является последовательность

$$\frac{0}{1}, \frac{1}{7}, \frac{1}{6}, \frac{1}{5}, \frac{1}{4}, \frac{2}{7}, \frac{1}{3}, \frac{2}{5}, \frac{3}{7}, \frac{1}{2}, \frac{4}{7}, \frac{3}{5}, \frac{2}{3}, \frac{5}{7}, \frac{3}{4}, \frac{4}{5}, \frac{5}{6}, \frac{6}{7}, \frac{1}{1}.$$

Если обозначить члены этого ряда через x_0/y_0, x_1/y_1, x_2/y_2, ..., то из упр. 19 следует, что

$$x_0 = 0, \quad y_0 = 1, \qquad x_1 = 1, \quad y_1 = n;$$
$$x_{k+2} = \lfloor (y_k + n)/y_{k+1} \rfloor x_{k+1} - x_k,$$
$$y_{k+2} = \lfloor (y_k + n)/y_{k+1} \rfloor y_{k+1} - y_k.$$

Напишите подпрограмму для MIX, которая вычисляет ряд Фарея порядка n, сохраняя значения x_k и y_k в ячейках $\mathtt{X} + k$, $\mathtt{Y} + k$ соответственно (Общее количество членов этого ряда приблизительно равно $3n^2/\pi^2$, поэтому можно предполагать, что n достаточно мало́)

19. [*M30*] (a) Покажите, что числа x_k и y_k, которые определяются рекуррентным соотношением из предыдущего упражнения, удовлетворяют равенству $x_{k+1}y_k - x_k y_{k+1} = 1$. (b) На основании факта, доказанного в п. (a), покажите, что дроби x_k/y_k действительно являются членами ряда Фарея порядка n

▸ **20.** [*33*] Предположим, что флаг переполнения и регистр X машины MIX подключены к светофору, расположенному на углу проспекта Дель-Мар (Del Mar Boulevard) и Беркли-авеню (Berkeley Avenue), следующим образом:

rX(2:2) = светофор для транспорта на Дель-Мар
rX(3:3) = светофор для транспорта на Беркли
} 0 — выключен, 1 — зеленый, 2 — желтый, 3 — красный;

rX(4.4) = сигнал для пешеходов на Дель-Мар
rX(5·5) = сигнал для пешеходов на Беркли
} 0 — выключен, 1 — "ИДИТЕ", 2 — "СТОЙТЕ".

Машины или пешеходы, которые хотят, двигаясь по Беркли, пересечь проспект, должны включить переключатель, который установит флаг переполнения MIX в положение 1 Если такая ситуация не возникнет, то сигнал светофора для Дель-Мар всегда будет оставаться зеленым.

При этом установлены следующие временные интервалы.

Зеленый свет для транспорта на Дель-Мар ≥ 30 с, желтый — 8 с.
Зеленый свет для транспорта на Беркли — 20 с, желтый — 5 с.

Когда в одном направлении для транспорта горит зеленый или желтый сигнал светофора, в другом направлении горит красный свет. Когда транспорту дан зеленый свет, то включен соответствующий сигнал СТОЙТЕ, только перед переключением зеленого света на желтый сигнал СТОЙТЕ мигает в течение 12 с по следующей схеме циклов:

$$\left.\begin{array}{ll}\texttt{СТОЙТЕ} & \frac{1}{2}\text{ с} \\ \text{Отключен} & \frac{1}{2}\text{ с}\end{array}\right\}\ \text{повторяется 8 раз;}$$

СТОЙТЕ 4 с (и остается во время циклов желтого и красного света).

Если флаг переполнения включился, когда на Беркли горит зеленый сигнал, то машина или пешеход пройдет в этом же цикле, но если это случилось во время цикла желтого или красного света, то придется ждать следующего цикла, который наступит после того, как проедет поток машин по Дель-Мар.

Пусть единица времени для MIX составляет 10 μsec. Напишите полную программу для MIX, которая управляет сигналами светофора с помощью rX, в зависимости от того, что подано на вход флага переполнения. Необходимо неукоснительно придерживаться установленных промежутков времени; исключение может быть сделано только для тех случаев, когда это невозможно. *Замечание.* Значение в rX меняется точно в момент *завершения* выполнения команды LDX или INCX.

21. [*28*] *Магический квадрат порядка* n — это такое расположение в квадрате чисел от 1 до n^2, при котором сумма элементов и по вертикали, и по горизонтали равна $n(n^2+1)/2$; такой же результат дает сумма элементов двух главных диагоналей. На рис. 16 показан магический квадрат порядка 7. Для его составления используется следующее правило. Начните с 1, вставив ее в клетку, расположенную прямо под "центральной" клеткой (см. рис. 16), затем двигайтесь вниз и вправо по диагонали (при выходе за границу квадрата представляйте себе, что вся плоскость выложена подобными квадратами), пока не достигнете заполненной клетки; затем опуститесь вниз на две клетки от последней заполненной клетки и продолжайте движение по диагонали вниз и вправо. Этот метод подходит для любого нечетного n.

Используя тот же принцип распределения памяти, что и в упр. 10, напишите полную программу для MIX, которая генерирует магический квадрат размера 23×23 описанным выше методом и печатает результат. [Этот алгоритм принадлежит Мануэлю Мосхопулосу (Manuel Moschopoulos), который жил в Константинополе приблизительно в 1300 году. Другие многочисленные и не менее интересные методы построения магических квадратов, которые представляют собой хорошие упражнения по программированию, можно найти в книге W. W. Rouse Ball, *Mathematical Recreations and Essays*, revised by H. S. M. Coxeter (New York: Macmillan, 1939), Chapter 7.]

22. [*31*] (*Задача Иосифа.*) Пусть n человек стоят по кругу. Начиная с некоторой позиции, они ведут отсчет по кругу и предают жестокой казни каждого m-го человека; после каждой казни круг смыкается. Например (рис. 17), при $n = 8$ и $m = 4$ казнить будут в следующем порядке: 54613872. Первого человека казнят пятым, второго — четвертым и т. д. Напишите полную программу для MIX, которая распечатывает порядок выполнения казней для $n = 24$, $m = 11$. Постарайтесь придумать эффективный алгоритм, который быстро работает при больших n и m (однажды это может спасти вам жизнь). [*Ссылки.* W. Ahrens, *Mathematische Unterhaltungen und Spiele* **2** (Leipzig: Teubner, 1918), Chapter 15.]

22	47	16	41	10	35	04
05	23	48	17	42	11	29
30	06	24	49	18	36	12
13	31	07	25	43	19	37
38	14	32	01	26	44	20
21	39	08	33	02	27	45
46	15	40	09	34	03	28

Рис. 16. Магический квадрат.

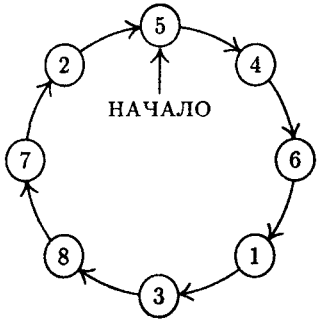


Рис. 17. Задача Иосифа, $n = 8$, $m = 4$.

23. [*37*] Цель этого упражнения — помочь читателю получить опыт применения компьютеров в сфере, где выходные данные должны быть отображены графически, а не в традиционном табличном виде. В данном случае необходимо "нарисовать" схему кроссворда.

Входными данными является матрица, состоящая из нулей и единиц. Нуль соответствует белому квадратику, а единица — черному. В качестве выходных данных нужно получить схему кроссворда, в котором соответствующие квадратики с номерами обозначают слова, расположенные по вертикали и по горизонтали.

Например, для матрицы

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

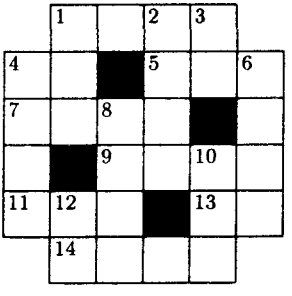


Рис. 18. Схема кроссворда, соответствующая матрице из упр. 23.

соответствующая схема кроссворда должна выглядеть, как показано на рис. 18. Квадратик нумеруется, если он белый и либо (a) под ним расположен белый квадратик и прямо над ним нет белого квадратика, либо (b) справа находится белый квадрат и непосредственно слева нет белого квадрата. Если черный квадратик оказался у края, то его нужно удалить из схемы. Это проиллюстрировано на рис. 18, где черные квадратики по углам были удалены. Существует простой способ достижения этой цели — "искусственно" вставить строки и столбцы, содержащие -1 сверху, снизу и по боковым сторонам заданной входной матрицы, а затем поменять каждую $+1$, которая соседствует с -1, на -1, пока рядом с любой -1 не останется ни одной $+1$.

Для печати окончательной схемы на АЦПУ следует использовать следующий метод. Каждый квадратик кроссворда должен отображаться на печатной странице с помощью 5 столбцов и 3 строк, причем эти 15 позиций заполняются следующим образом.

Непронумерованные	␣␣␣␣+	Номер nn	nn␣␣+	Черные	+++++
белые квадраты:	␣␣␣␣+	белого квадрата:	␣␣␣␣+	квадраты:	+++++
	+++++		+++++		+++++

Квадраты с -1 в зависимости от того, справа или снизу находятся -1:

␣␣␣␣+	␣␣␣␣+	␣␣␣␣␣	␣␣␣␣␣	␣␣␣␣␣
␣␣␣␣+	␣␣␣␣+	␣␣␣␣␣	␣␣␣␣␣	␣␣␣␣␣
+++++	␣␣␣␣+	+++++	␣␣␣␣+	␣␣␣␣␣

Схема, изображенная на рис. 18, будет напечатана, как показано на рис. 19.

Ширины строки принтера — 120 символов — достаточно для того, чтобы печатать кроссворды, содержащие до 23 столбцов. В качестве входных данных должна быть предоставлена матрица размера 23 × 23, состоящая из нулей и единиц, все строки которой перфорируются в столбцах 1–23 входной перфокарты. Например, карта, соответствующая первой строке приведенной выше матрицы, будет перфорирована следующим образом: "10000111111111111111111". Схема кроссворда необязательно будет симметричной, и у нее могут оказаться длинные ленты черных квадратиков, присоединенные к наружным сторонам кроссворда самым экзотическим образом.

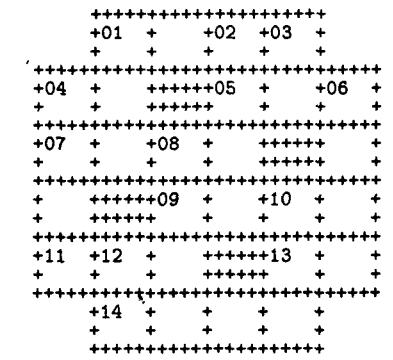


Рис. 19. Распечатка на АЦПУ кроссворда, показанного на рис. 18.

1.3.3. Применение к перестановкам

В данном разделе мы приведем еще несколько примеров программ для MIX и наряду с этим познакомимся с некоторыми важными свойствами перестановок. Эти исследования позволят также выявить интересные аспекты компьютерного программирования в целом.

Перестановки уже обсуждались в разделе 1.2.5; в нем перестановка $c\,d\,f\,b\,e\,a$ рассматривалась как некоторое *расположение* шести объектов a, b, c, d, e, f в ряд. Но возможна и другая точка зрения. Перестановку можно рассматривать как *переупорядочение или переименование объектов*. При такой интерпретации* обычно используют двухстрочную запись, например

$$\begin{pmatrix} a & b & c & d & e & f \\ c & d & f & b & e & a \end{pmatrix}. \tag{1}$$

Это означает, что "a переходит в c, b переходит в d, c переходит в f, d переходит в b, e переходит в e, f переходит в a". С точки зрения переупорядочения это означает, что объект c переходит на место, которое ранее занимал объект a. А если рассматривать это как переименование, то можно считать, что объект a переименован в c. При использовании двухстрочной записи изменение порядка столбцов в перестановке никакой роли не играет. Например, перестановку (1) можно записать в виде

$$\begin{pmatrix} c & d & f & b & a & e \\ f & b & a & d & c & e \end{pmatrix},$$

а также еще 718 способами.

В связи с описанной интерпретацией часто используют *циклическую запись*. Перестановку (1) можно записать и в виде

$$(a\,c\,f)\,(b\,d), \tag{2}$$

---

* В русскоязычной математической литературе в этом случае часто используют термин "подстановка". — *Прим. перев.*

что, опять же, означает "a переходит в c, c переходит в f, f переходит в a, b переходит в d, d переходит в b". Цикл $(x_1\, x_2 \ldots x_n)$ означает "x_1 переходит в x_2, $\ldots$, x_{n-1} переходит в x_n, x_n переходит в x_1". Так как элемент e является в перестановке фиксированным (т. е. переходит не в какой-либо другой элемент, а в самого себя), он не появляется в циклической записи. Таким образом, единичные циклы типа "(e)" записывать не принято. Если в перестановке фиксированными *являются все элементы, так что присутствуют только единичные циклы, то она* называется *тождественной перестановкой* и обозначается "()".

Запись перестановки в виде цикла не является единственной. Например,

$$(b\,d)\,(a\,c\,f), \qquad (c\,f\,a)\,(b\,d), \qquad (d\,b)\,(f\,a\,c) \tag{3}$$

и т. д. эквивалентны (2). Но запись "$(a\,f\,c)\,(b\,d)$" уже не будет им эквивалентна, так как она означает, что a переходит в f.

Легко видеть, почему перестановку всегда можно представить в виде цикла. Начиная с элемента x_1, перестановка переводит, скажем, x_1 в x_2, x_2 в x_3 и т. д., пока наконец (поскольку количество элементов ограничено) мы не придем к некоему элементу x_{n+1}, который уже встречался среди $x_1, \ldots, x_n$. Этот элемент x_{n+1} должен быть равен x_1. Предположим, он равен x_3. Но это невозможно, так как известно, что x_2 переходит в x_3, а по предположению в x_{n+1} переходит $x_n \neq x_2$. Поэтому $x_{n+1} = x_1$ и получаем цикл $(x_1\, x_2 \ldots x_n)$, являющийся частью перестановки для некоторого $n \geq 1$. Если этим циклом вся перестановка не исчерпывается, то существует элемент y_1, такой, что $y_1 \neq x_i$ для всех i, для которого аналогичным образом получим еще один цикл $(y_1\, y_2 \ldots y_m)$. Ни один y_i не может равняться любому x_i, так как из $x_i = y_j$ следует, что $x_{i+1} = y_{j+1}$ и т. д. В конце концов для некоторого k получим $x_k = y_1$, что противоречит предположению о выборе y_1. Действуя подобным образом, можно найти все циклы, содержащиеся в перестановке.

В программировании эти понятия применяются каждый раз, когда некоторый набор из n объектов нужно расположить в другом порядке. Чтобы переупорядочить объекты, не перемещая их в какое-либо другое место, необходимо, в сущности, придерживаться циклической структуры. Например, чтобы переупорядочить (1), т. е. присвоить

$$(a,\, b,\, c,\, d,\, e,\, f) \leftarrow (c,\, d,\, f,\, b,\, e,\, a),$$

будем следовать циклической структуре (2) и последовательно присваивать

$$t \leftarrow a, \quad a \leftarrow c, \quad c \leftarrow f, \quad f \leftarrow t; \quad t \leftarrow b, \quad b \leftarrow d, \quad d \leftarrow t.$$

Но нужно отдавать себе отчет в том, что любое подобное преобразование можно осуществлять в непересекающихся циклах.

Произведения перестановок. Две перестановки можно перемножить в том смысле, что их произведение означает применение одной перестановки вслед за другой. Например, если за перестановкой (1) следует перестановка

$$\begin{pmatrix} a & b & c & d & e & f \\ b & d & c & a & f & e \end{pmatrix}$$

то получим, что a переходит в c, которое затем переходит в c; b переходит в d, которое затем переходит в a и т. д.:

$$\begin{pmatrix} a & b & c & d & e & f \\ c & d & f & b & e & a \end{pmatrix} \times \begin{pmatrix} a & b & c & d & e & f \\ b & d & c & a & f & e \end{pmatrix}$$
$$= \begin{pmatrix} a & b & c & d & e & f \\ c & d & f & b & e & a \end{pmatrix} \times \begin{pmatrix} c & d & f & b & e & a \\ c & a & e & d & f & b \end{pmatrix}$$
$$= \begin{pmatrix} a & b & c & d & e & f \\ c & a & e & d & f & b \end{pmatrix}. \tag{4}$$

Вполне очевидно, что произведение перестановок не обладает свойством коммутативности; другими словами, $\pi_1 \times \pi_2$ необязательно равно $\pi_2 \times \pi_1$, где π_1 и π_2 — перестановки. Читатель может убедиться в том, что, если поменять местами сомножители, произведение в (4) даст другой результат (см. упр. 3).

Кое-кто перемножает перестановки справа налево, вместо того чтобы делать это более естественным образом слева направо, как показано в (4). Фактически математики разделились на два лагеря: одни считают, что результат последовательного применения преобразований T_1 и T_2 следует обозначать через T_1T_2, а другие — через T_2T_1. Здесь мы будем использовать обозначение T_1T_2.

Пользуясь циклической записью, запишем равенство (4) следующим образом:

$$(a\,c\,f)\,(b\,d)\,(a\,b\,d)\,(e\,f) = (a\,c\,e\,f\,b). \tag{5}$$

Обратите внимание на то, что знак умножения "$\times$" принято опускать; это не противоречит циклической записи, так как легко видеть, что перестановка $(a\,c\,f)(b\,d)$ на самом деле является произведением перестановок $(a\,c\,f)$ и $(b\,d)$.

Произведение перестановок можно выполнять непосредственно с помощью циклической записи. Например, вычислив произведение перестановок

$$(a\,c\,f\,g)\,(b\,c\,d)\,(a\,e\,d)\,(f\,a\,d\,e)(b\,g\,f\,a\,e), \tag{6}$$

находим (двигаясь слева направо), что "a переходит в c, c переходит в d, d переходит в a, a переходит в d и d остается без изменений". Таким образом, конечным результатом (6) является то, что a переходит в d, и мы запишем "$(a\,d$" в качестве частичного ответа. Теперь выясним, что происходит с d: "d переходит в b, которое затем переходит в g"; следовательно, имеем частичный результат "$(a\,d\,g$". Рассмотрев, что происходит с g, находим, что "g переходит в a, затем в e, в f и в a"; таким образом, первый цикл замыкается: "$(a\,d\,g)$". Теперь выберем новый элемент, который еще не встречался, например c. Находим, что c переходит в e, и читатель может проверить, что окончательным ответом для (6) будет "$(a\,d\,g)(c\,e\,b)$".

А теперь попробуем выполнить эту процедуру с помощью компьютера. Следующий алгоритм формализует описанный выше метод таким образом, чтобы его можно было выполнить с помощью компьютера.

Алгоритм А (*Перемножение перестановок, представленных в виде циклов*). Этот алгоритм (рис. 20) берет произведение циклов, такое как (6), и вычисляет получающуюся в результате перестановку в виде произведения непересекающихся циклов. Для простоты здесь не приводится описание процедуры удаления единичных циклов; это было бы лишь незначительным обобщением алгоритма. По мере выпол-

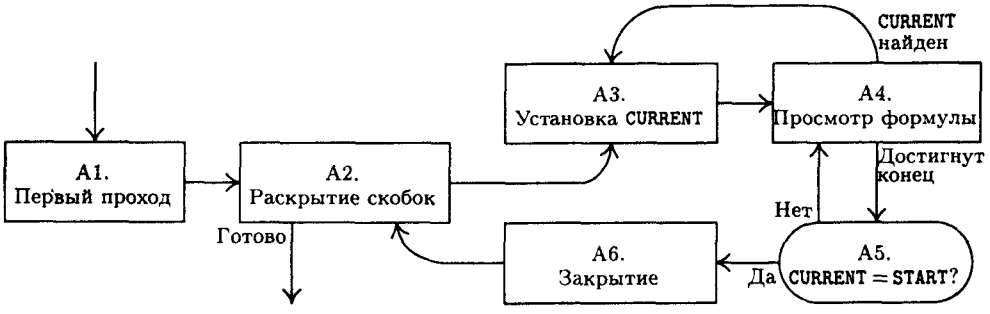


Рис. 20. Алгоритм A (перемножение перестановок).

нения алгоритма последовательно "помечаем" элементы исходной формулы, т. е. те символы, которые уже "обработаны".

A1. [Первый проход.] Отметить все левые скобки и заменить все правые скобки помеченной копией входного символа, который следует за соответствующей левой скобкой (см. пример в табл. 1).

A2. [Раскрытие скобок.] Выполнив поиск слева направо, найти первый непомеченный элемент входной формулы. (Если все элементы помечены, то работа алгоритма заканчивается.) Установить START равным этому элементу; вывести левую скобку; вывести элемент и пометить его.

A3. [Установка CURRENT.] Установить CURRENT равным следующему элементу формулы.

A4. [Просмотр формулы.] Продвигаться вправо, пока не будет либо достигнут конец формулы, либо найден элемент, равный CURRENT; в последнем случае пометить его и вернуться к шагу A3.

A5. [CURRENT = START?] Если CURRENT $\neq$ START, вывести CURRENT и вернуться к шагу A4, снова начав с левого края формулы (продолжая тем самым вывод искомого цикла).

A6. [Закрытие.] (Полный искомый цикл выведен.) Вывести правую скобку и вернуться к шагу A2. ∎

Например, рассмотрим формулу (6). В табл. 1 показаны последовательные этапы ее обработки. В первой строке таблицы приведена формула после замены правых скобок начальными элементами соответствующих циклов. В последующих строках показан процесс обработки формулы по мере пометки все большего числа элементов. Курсор указывает на текущий элемент, подлежащий обработке. В результате будет выведено следующее выражение: "$(a\,d\,g)(c\,e\,b)(f)$". Обратите внимание, что в выводе присутствует единичный цикл.

Программа для MIX. Чтобы реализовать этот алгоритм для MIX, "пометки" можно делать с помощью знака слова. Предположим, что входная формула перфорирована на картах в следующем формате: состоящая из 80 колонок карта разделена на 16 полей по 5 символов. Каждое поле представляет собой один из следующих вариантов: (a) "␣␣␣␣(", что обозначает левую скобку начала цикла; (b) ")␣␣␣␣",

Таблица 1

ПРИМЕНЕНИЕ АЛГОРИТМА А К ВЫРАЖЕНИЮ (6)

После шага	START	CURRENT	(a c f g)	(b c d)	(a e d)	(f a d e)	(b g f a e)	Вывод
A1			a c f g	b c d	a e d	f a d e	b g f a e	
A2	a		⌶c f g	b c d	a e d	f a d e	b g f a e	(a
A3	a	c	c⌶f g	b c d	a e d	f a d e	b g f a e	
A4	a	c	c f g	b ⌶d	a e d	f a d e	b g f a e	
A4	a	d	c f g	b d	a e	f a d e	b g f a e	
A4	a	a	c f g	b d	a e	f ⌶d e	b g f a e	
A5	a	d	c f g	b d	a e	f d e	b g f a e ⌶	d
A5	a	g	c f g	b	a e	f d e	g f a e ⌶	g
A5	a	a	c f	b	e	f d	g a e ⌶	
A6	a	a	c f	b	e	f d	g a e ⌶	)
A2	c	a	⌶f	b	e	f d	g a e	(c
A5	c	e	f	b	e	d	g e ⌶	e
A5	c	b	f	b			g ⌶	b
A6	c	c	f				g ⌶	)
A6	f	f					⌶	(f)

Здесь ⌶ обозначает положение курсора сразу за только что проверенным элементом помеченные элементы выделены светло серым цветом

что обозначает правую скобку конца цикла, (c) "␣␣␣␣␣", т е все пробелы, которые можно вставить в любом месте, чтобы заполнить пространство, (d) любой символ, обозначающий элемент, который нужно переставить Последняя карта входных данных определяется по тому, что в ее колонках 76–80 содержится "␣␣␣␣=" Например, (6) можно перфорировать на двух картах следующим образом

```
   ( A   C   F   G )     ( B   C   D )       ( A   E   D )
   ( F   A   D   E )     ( B   G   F   A   E )                 =
```

В выводе нашей программы будет содержаться точная копия ввода с последующим ответом в таком же, в сущности, формате

Программа А (*Перемножение перестановок, представленных в виде циклов*) В этой программе реализован алгоритм А, а также предусмотрены процедуры ввода, вывода и удаления единичных циклов

```
01 MAXWDS  EQU  1200          Максимальная длина ввода
02 PERM    ORIG *+MAXWDS      Вводная перестановка
03 ANS     ORIG *+MAXWDS      Место для ответа
04 OUTBUF  ORIG *+24          Место для печати
05 CARDS   EQU  16            Номер устройства чтения перфокарт
06 PRINTER EQU  18            Номер АЦПУ
07 BEGIN   IN   PERM(CARDS)   Читать первую карту
```

```
08          ENT2 0
09          LDA  EQUALS
10  1H      JBUS *(CARDS)          Ожидать окончания цикла
11          CMPA PERM+15,2
12          JE   *+2               Это последняя перфокарта?
13          IN   PERM+16,2(CARDS)  Нет, читать следующую.
14          ENT1 OUTBUF
15          JBUS *(PRINTER)        Напечатать копию введенной
16          MOVE PERM,2(16)          перфокарты.
17          OUT  OUTBUF(PRINTER)
18          JE   1F
19          INC2 16
20          CMP2 =MAXWDS-16=
21          JLE  1B                Повторять до окончания ввода.
22          HLT  666               Слишком много входной информации!
23  1H      INC2 15                1  В данный момент rI2 введенных слов
24          ST2  SIZE              1    находятся в ячейках PERM, PERM + 1, ....
25          ENT3 0                 1  A1. Первый проход.
26  2H      LDAN PERM,3            A  Взять следующий элемент ввода.
27          CMPA LPREN(1:5)        A  Это "("?
28          JNE  1F                A
29          STA  PERM,3            B  Если да, пометить ее.
30          INC3 1                 B  Поместить следующий непустой символ ввода
31          LDXN PERM,3            B    в rX
32          JXZ  *-2               B
33  1H      CMPA RPREN(1:5)        C
34          JNE  *+2               C
35          STX  PERM,3            D  Заменить ")" помеченным элементом из rX.
36          INC3 1                 C
37          CMP3 SIZE              C  Все элементы обработаны?
38          JL   2B                C
39          LDA  LPREN             1  Подготовиться к главной программе.
40          ENT1 ANS               1  rI1 — место сохранения следующего ответа.
41  OPEN    ENT3 0                 E  A2. Открытие.
42  1H      LDXN PERM,3            F  Искать непомеченный элемент
43          JXN  GO                F
44          INC3 1                 G
45          CMP3 SIZE              G
46          JL   1B                G
47  *                                 Все элементы помечены. Выполняется вывод
48  DONE    CMP1 =ANS=
49          JNE  *+2                  Ответ является тождественной перестановкой?
50          MOVE LPREN(2)             Если да, заменить на "()"
51          MOVE =0=                  Поместить 23 слова пробелов после ответа.
52          MOVE -1,1(22)
53          ENT3 0
54          OUT  ANS,3(PRINTER)
55          INC3 24
56          LDX  ANS,3                Напечатать столько строк, сколько необходимо.
57          JXNZ *-3
```

```
58          HLT
59  *
60  LPREN   ALF      (                Константы, используемые в программе.
61  RPREN   ALF   )
62  EQUALS  ALF      =
63  *
64  GO      MOVE LPREN          H    Открыть цикл в выводе.
65          MOVE PERM,3         H
66          STX  START          H
67  SUCC    STX  PERM,3         J    Пометить элемент.
68          INC3 1              J    Сделать один шаг вправо.
69          LDXN PERM,3(1:5)    J    A3. Установить CURRENT (а именно — rX).
70          JXN  1F             J    Пропустить пробелы.
71          JMP  *-3            0
72  5H      STX  0,1            Q    Вывести CURRENT.
73          INC1 1              Q
74          ENT3 0              Q    Снова просмотреть формулу.
75  4H      CMPX PERM,3(1:5)    K    A4. Просмотр формулы.
76          JE   SUCC           K    Элемент = CURRENT?
77  1H      INC3 1              L    Продвинуться вправо.
78          CMP3 SIZE           L    Конец формулы?
79          JL   4B             L
80          CMPX START(1:5)     P    A5. CURRENT = START?
81          JNE  5B             P
82  CLOSE   MOVE RPREN          R    A6. Закрытие.
83          CMPA -3,1           R    Заметить: rA = "(".
84          JNE  OPEN           R
85          INC1 -3             S    Удалить единичные циклы.
86          JMP  OPEN           S
87          END  BEGIN
```

Эта программа, содержащая примерно 75 команд, значительно длиннее программ из предыдущего раздела, да и большинства программ, с которыми мы встретимся в этой книге. Но ее длина не должна внушать вам страх, так как она делится на несколько маленьких частей, которые достаточно независимы одна от другой. В строках 07–22 происходит считывание входных перфокарт и печать копии каждой карты; в строках 23–38 выполняется шаг A1 алгоритма, т. е. предварительная обработка входных данных; в строках 39–46 и 64–86 выполняется основная часть алгоритма A, а в строках 48–57 выводится ответ. Читателю полезно изучить как можно больше программ для MIX, приведенных в данной книге, поскольку чрезвычайно важно приобрести опыт чтения программ, написанных другими. Но, к сожалению, подобной формой обучения пренебрегали на многих компьютерных курсах, что привело к крайне неэффективному использованию компьютерной техники.

Время выполнения. Те части программы A, которые не связаны с операциями ввода-вывода, снабжены счетчиками частоты, указывающими, сколько раз они выполняются (как в программе 1.3.2M); таким образом, предполагается, что строка 30 выполняется B раз. Для удобства считается, что во входной информации пустые слова могут находиться только у правого края; при этом условии никогда не выполняется строка 71 и никогда не происходит переход в строке 32.

Выполняя простые операции сложения, получаем, что общее время выполнения программы равно

$$(7 + 5A + 6B + 7C + 2D + E + 3F + 4G + 8H + 6J + 3K + 4L + 3P + 4Q + 6R + 2S)u \quad (7)$$

плюс время, затрачиваемое на ввод и вывод. Чтобы понять смысл формулы (7), нужно рассмотреть пятнадцать неизвестных A, B, C, D, E, F, G, H, J, K, L, P, Q, R, S и связать их с соответствующими характеристиками входных данных. Проиллюстрируем общие принципы решения проблем подобного рода.

Сначала применим первый закон Кирхгофа теории электрических цепей: команда должна выполняться столько раз, сколько раз осуществляется переход к ней. Это, казалось бы, очевидное правило часто приводит к неочевидным соотношениям между несколькими величинами. Анализируя ход выполнения программы A, получаем следующие уравнения.

Из строк	*Выведено*
26, 38	$A = 1 + (C - 1)$
33, 28	$C = B + (A - B)$
41, 84, 86	$E = 1 + R$
42, 46	$F = E + (G - 1)$
64, 43	$H = F - G$
67, 70, 76	$J = H + (K - (L - J))$
75, 79	$K = Q + (L - P)$
82, 72	$R = P - Q$

Не все уравнения, полученные с помощью закона Кирхгофа, будут линейно независимыми, например в данном случае мы видим, что первое и второе уравнения явно эквивалентны. Более того, последнее уравнение можно вывести из остальных, так как из третьего, четвертого и пятого следует, что $H = R$. Таким образом, шестое означает, что $K = L - R$. Во всяком случае мы уже исключили шесть из пятнадцати неизвестных:

$$A = C, \quad E = R + 1, \quad F = R + G, \quad H = R, \quad K = L - R, \quad Q = P - R. \quad (8)$$

Первый закон Кирхгофа — это очень эффективное средство; более подробно мы рассмотрим его в разделе 2.3.4.1.

Следующий шаг состоит в том, чтобы попытаться сопоставить переменные с важными характеристиками данных. Из строк 24, 25, 30 и 36 находим, что

$$B + C = \text{количество слов ввода} = 16X - 1, \quad (9)$$

где X — число перфокарт с входной информацией. Из строки 28 получаем, что

$$B = \text{количество “(” во вводе} = \text{количество циклов во вводе.} \quad (10)$$

Аналогично из строки 34 получаем

$$D = \text{количество “)” во вводе} = \text{количество циклов во вводе.} \quad (11)$$

А теперь из (10) и (11) получаем то, что нельзя вывести из закона Кирхгофа:

$$B = D. \tag{12}$$

Из строки 64 получаем

$$H = \text{количество циклов в выводе (включая единичные циклы).} \tag{13}$$

Из строки 82 следует, что R равно этой же величине; в данном случае соотношение $H = R$ *можно было* вывести из закона Кирхгофа, так как оно уже содержится в (8).

Учитывая тот факт, что каждое непустое слово в конце концов помечается, из строк 29, 35 и 67 находим, что

$$J = Y - 2B, \tag{14}$$

где Y — число непустых слов, содержащихся в перестановках ввода. Учитывая то, что каждый *отличный от других* элемент, содержащийся во входной перестановке, записывается в вывод только один раз, либо в строке 65, либо в строке 72, получаем

$$P = H + Q = \text{количество различных элементов во вводе.} \tag{15}$$

(См. соотношения (8).) Если немного поразмыслить, это очевидным образом следует также из строки 80. И, наконец, из строки 85 видно, что

$$S = \text{количество единичных циклов в выводе.} \tag{16}$$

Очевидно. что величины B, C, H, J, P и S, которые мы сейчас интерпретировали как независимые параметры, влияют на время выполнения программы A.

Теперь остается проанализировать только неизвестные G и L. Для этого придется проявить немного больше изобретательности. Просмотры входной информации, которые начинаются в строках 41 и 74, всегда заканчиваются либо в строке 47 (прошлый раз), либо в строке 80. В течение каждого из этих $P + 1$ циклов команда "`INC3 1`" выполняется $B + C$ раз. Это имеет место только в строках 44, 68 и 77, поэтому получаем нетривиальное соотношение

$$G + J + L = (B + C)(P + 1), \tag{17}$$

связывающее неизвестные G и L. К счастью, время выполнения (7) — это функция от $G + L$ (так как оно включает слагаемые $\cdots + 3F + 4G \cdots + 3K + 4L + \cdots = \cdots + 7G + \cdots + 7L + \cdots$), поэтому нам не нужно больше пытаться анализировать величины G и L по отдельности.

Просуммировав все эти результаты, находим, что общее время выполнения программы без учета времени, затрачиваемого на операции ввода-вывода, равно

$$(112NX + 304X - 2M - Y + 11U + 2V - 11)u. \tag{18}$$

В этой формуле были использованы новые обозначения для характеристик данных:

$$\begin{aligned}
&X \text{ — количество перфокарт ввода;}\\
&Y \text{ — количество непустых полей во вводе (исключая конечное "=");}\\
&M \text{ — количество циклов во вводе;}\\
&N \text{ — количество имен различных элементов во вводе;}\\
&U \text{ — количество циклов в выводе (включая единичные циклы);}\\
&V \text{ — количество единичных циклов в выводе.}
\end{aligned} \tag{19}$$

Таблица 2

ПЕРЕМНОЖЕНИЕ ПЕРЕСТАНОВОК ЗА ОДИН ПРОХОД

	$(a\ f\ g)\ (b\ c\ d)\ (a\ e\ d)\ (f\ a\ d\ e)\ (b\ g\ f\ a\ e)$
$a \to$	$d\,d\,a\,a\,a\,a\,a\,a\,a\,a\,a\,a\,a\,d\,d\,d\,d\,d\,d\,e\,e\,e\,e\,e\,e\,e\,e\,e\,a\,a$
$b \to$	$c\,c\,c\,c\,c\,c\,c\,c\,c\,g\,g\,g\,g\,g\,g\,g\,g\,g\,g\,g\,g\,g\,g\,g\,b\,b\,b\,b\,b$
$c \to$	$e\,e\,e\,d\,d\,d\,d\,d\,d\,c\,c\,c\,c\,c\,c\,c\,c\,c\,c\,c\,c\,c\,c\,c\,c\,c\,c\,c\,c\,c$
$d \to$	$g\,g\,g\,g\,g\,g\,g\,)\,)\,)\,d\,d\,)\,)\,)\,b\,b\,b\,b\,b\,d\,d\,d\,d\,d\,d\,d\,d\,d$
$e \to$	$b\,b\,b\,b\,b\,b\,b\,b\,b\,b\,b\,b\,b\,b\,a\,a\,a\,)\,)\,)\,)\,b\,b\,)\,)\,)\,)\,)\,e$
$f \to$	$f\,f\,f\,f\,e\,e\,e\,e\,e\,e\,e\,e\,e\,e\,e\,e\,e\,e\,e\,a\,a\,a\,a\,a\,a\,a\,a\,f\,f\,f$
$g \to$	$a\,)\,)\,)\,)\,f\,f\,f\,f\,f\,f\,f\,f\,f\,f\,f\,f\,f\,f\,f\,f\,f\,f\,f\,f\,g\,g\,g\,g$

Следовательно, можно сделать вывод, что анализ такой программы, как A, во многих отношениях подобен решению занимательной головоломки.

Ниже мы покажем,что если на выходе предполагается получить случайную перестановку, то средние величин U и V будут равны H_N и 1 соответственно.

Другой подход. Алгоритм A перемножает перестановки почти так же, как это обычно делают люди. Часто оказывается, что задачи, решаемые с помощью компьютера, очень похожи на те, с которыми люди постоянно сталкивались в течение долгих лет. Поэтому освященные веками методы решения, разработанные для простых смертных, таких, как мы с вами, подходят и для компьютерных алгоритмов.

Но не менее часто приходится иметь дело с новыми методами, которые идеально подходят для компьютера, но совершенно непригодны для человека. Главная причина этого состоит в том, что компьютер "думает" по-другому; тип его памяти, с помощью которой он запоминает информацию, отличается от типа памяти человека. Это различие можно проследить на примере нашей задачи перемножения перестановок. Используя приведенный ниже алгоритм, компьютер может выполнить умножение перестановок "одним махом", запоминая текущее состояние перестановки в целом по мере перемножения циклов. Предназначенный для человека алгоритм A просматривает формулу много раз, по одному для каждого элемента вывода, в то время как новый алгоритм делает все за один проход. *Homo sapiens* на такое вряд ли способен.

Что представляет собой предназначенный для компьютера метод перемножения перестановок? Главная идея проиллюстрирована в табл. 2. В столбце, расположенном в этой таблице под каждым символом выражения, которое представлено в виде циклов, показано, какая перестановка выражена частичными циклами *справа*. Например, фрагмент формулы "... $d\,e)(b\ g\ f\ a\ e)$" представляет перестановку

$$\begin{pmatrix} a & b & c & d & e & f & g \\ e & g & c & b & ? & a & f \end{pmatrix},$$

которая появляется в таблице под крайним справа символом d.

Внимательно изучив табл. 2, можно понять принцип ее построения, если начать с тождественной перестановки справа и двигаться назад справа налево Столбец, расположенный под символом x, отличается от столбца справа (в котором записано

предыдущее состояние) только строкой x; и новое значение в строке x — это значение, которое исчезло в результате предыдущего изменения. Короче говоря, имеем следующий алгоритм.

Алгоритм B (*Перемножение перестановок, представленных в виде циклов*). Этот алгоритм (рис. 21) дает, в сущности, тот же результат, что и алгоритм A. Предположим, что переставляемыми элементами являются $x_1, x_2, \ldots, x_n$. Будем использовать вспомогательную таблицу $T[1], T[2], \ldots, T[n]$; по окончании работы алгоритма x_i переходит в x_j в перестановке ввода тогда и только тогда, когда $T[i] = j$.

B1. [Инициализация.] Присвоить $T[k] \leftarrow k$, где $1 \le k \le n$. Подготовиться также к просмотру справа налево.

B2. [Следующий элемент.] Рассмотреть следующий элемент ввода (справа налево). Если ввод исчерпан, то работа алгоритма заканчивается. Если рассматривается элемент ")", то присвоить $Z \leftarrow 0$ и повторить шаг B2; если это элемент "(", то перейти к шагу B4. В противном случае рассматривается элемент x_i для некоторого i; тогда перейти к шагу B3.

B3. [Замена $T[i]$.] Выполнить взаимный обмен $Z \leftrightarrow T[i]$. Если в результате получится $T[i] = 0$, то присвоить $j \leftarrow i$. Вернуться к шагу B2.

B4. [Замена $T[j]$.] Присвоить $T[j] \leftarrow Z$. (Здесь j — это строка, определяющая элемент ")" в обозначениях табл. 2, т. е. правую скобку, которая соответствует только что просмотренной левой скобке.) Вернуться к шагу B2. ▮

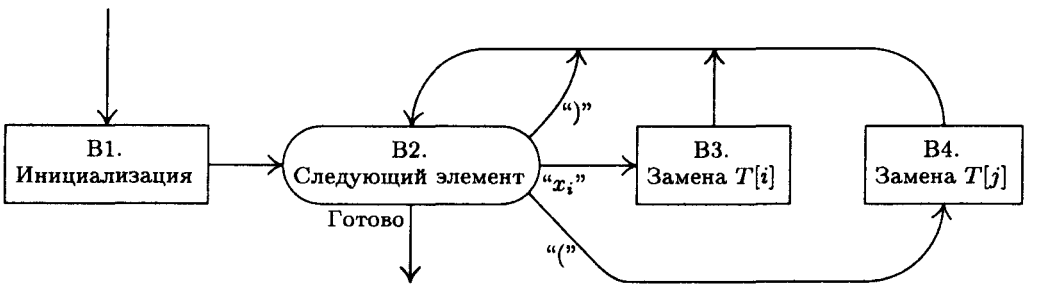


Рис. 21. Алгоритм B: перемножение перестановок.

Разумеется, после выполнения этого алгоритма необходимо еще вывести содержимое таблицы T в циклическом виде; как будет показано ниже, это легко сделать методом "пометок".

Теперь на основе нового алгоритма давайте напишем программу для MIX. Будем придерживаться тех же основных правил, что и в программе A, используя такой же формат ввода и вывода, как и раньше. Но тут возникает небольшая проблема: как можно реализовать алгоритм B, не зная наперед, чему равны элементы $x_1, x_2, \ldots, x_n$? Неизвестно, чему равно n, а также будет ли элемент b равен x_1 или x_2 и т. д. Существует простой способ решения данной проблемы — создать таблицу, внести в нее имена элементов, которые уже встречались, и каждый раз искать имя текущего элемента (см. строки 35–44 в программе ниже).

Программа B (*Дает тот же результат, что и программа A*). rX $\equiv Z$; rI4 $\equiv i$; rI1 $\equiv j$; rI3 $= n$, количество различных просмотренных имен.

```
01  MAXWDS EQU  1200          Максимальная длина ввода.
02  X      ORIG *+MAXWDS      Таблица имен.
03  T      ORIG *+MAXWDS      Вспомогательная таблица состояния.
04  PERM   ORIG *+MAXWDS      Входная перестановка.
05  ANS    EQU  PERM          Место для ответа.
06  OUTBUF ORIG *+24          Место для печати.
07  CARDS  EQU  16      }
    ...                 }     Совпадают со строками 05–22 программы A.
24         HLT  666     }     В этот момент rI2 слов
25  1H     INC2 15        1      ввода находятся в PERM, PERM + 1, ...
26         ENT3 1         1      и мы пока еще не видели ни одного имени.
27  RIGHT  ENTX 0         A   Присвоить Z ← 0.
28  SCAN   DEC2 1         B   B2. Следующий элемент.
29         LDA  PERM,2    B
30         JAZ  CYCLE     B   Пропустить пробелы.
31         CMPA RPREN     C
32         JE   RIGHT     C   Следующим элементом является ")"?
33         CMPA LPREN     D
34         JE   LEFT      D   Это "("?
35         ENT4 1,3       E   Приготовиться к поиску.
36         STA  X         E   Сохранить в начале таблицы.
37  2H     DEC4 1         F   Искать в таблице имен.
38         CMPA X,4       F
39         JNE  2B        F   Повторять до нахождения соответствия.
40         J4P  FOUND     G   Это имя появлялось ранее?
41         INC3 1         H   Нет; увеличить размер таблицы.
42         STA  X,3       H   Вставить новое имя x_n.
43         ST3  T,3       H   Присвоить T[n] ← n,
44         ENT4 0,3       H     i ← n.
45  FOUND  LDA  T,4       J   B3. Замена T[i].
46         STX  T,4       J   Сохранить Z.
47         SRC  5         J   Установить Z.
48         JANZ SCAN      J
49         ENT1 0,4       K   Если Z было нулем, присвоить j ← i.
50         JMP  SCAN      K
51  LEFT   STX  T,1       L   B4. Замена T[j].
52  CYCLE  J2P  SCAN      P   Вернуться к B2, если работа еще не закончена.
53  *
54  OUTPUT ENT1 ANS       1   Весь ввод просмотрен.
55         J3Z  DONE      1   Таблицы x и T содержат ответ.
56  1H     LDAN X,3       Q   Теперь построим циклическую запись.
57         JAP  SKIP      Q   Имя было помечено?
58         CMP3 T,3       R   Это единичный цикл?
59         JE   SKIP      R
60         MOVE LPREN     S   Открыть цикл.
61  2H     MOVE X,3       T
62         STA  X,3       T   Пометить имя.
```

```
63          LD3  T,3        T  Найти элемент, в который переходит этот элемент.
64          LDAN X,3        T
65          JAN  2B         T  Он уже помечен?
66          MOVE RPREN      W  Да, цикл закрывается.
67    SKIP  DEC3 1          Z  Перейти к следующему имени.
68          J3P  1B         Z
69    *
70    DONE  CMP1 =ANS=   }
      ...                } Совпадают со строками 48–62 программы A.
84    EQUALS ALF     =   }
85          END  BEGIN      ▮
```

Строки 54–68, в которых строится циклическая запись на основании таблицы T и таблицы имен, представляют собой небольшой алгоритм, который заслуживает внимания. Величины A, B, ..., R, S, T, W, Z, влияющие на время выполнения программы, конечно, отличаются от одноименных величин, которые использовались при анализе программы A. Анализ этих величин будет интересным упражнением для читателя (см. упр. 10).

Опыт показывает, что основная часть времени выполнения программы B будет потрачена на поиск в таблице имен; это время определяется параметром F. Существуют более эффективные алгоритмы поиска и построения словарей имен; они называются *алгоритмами таблиц символов* и играют важную роль в прикладной математике. В главе 6 мы займемся всесторонним исследованием эффективных алгоритмов таблиц символов.

Обратные перестановки. Обратная перестановка π^- для перестановки π — это такая перестановка, которая отменяет действие π; если в π элемент i переходит в j π, то в π^- элемент j переходит в i. Таким образом, произведение $\pi\pi^-$ равняется тождественной перестановке; то же самое справедливо и для произведения $\pi^-\pi$. Обратную перестановку часто обозначают через π^{-1}, а не π^-, но верхний индекс 1 явно лишний (по той же причине, по которой $x^1 = x$).

Каждая перестановка имеет обратную. Например, обратной перестановкой для

$$\begin{pmatrix} a & b & c & d & e & f \\ c & d & f & b & e & a \end{pmatrix} \quad \text{является} \quad \begin{pmatrix} c & d & f & b & e & a \\ a & b & c & d & e & f \end{pmatrix} = \begin{pmatrix} a & b & c & d & e & f \\ f & d & a & b & e & c \end{pmatrix}.$$

Рассмотрим некоторые простые алгоритмы вычисления обратной перестановки.

Предположим, что в оставшейся части раздела мы будем иметь дело с перестановками чисел $\{1, 2, \ldots, n\}$. Если $X[1]X[2]\ldots X[n]$ — такая перестановка, то существует простой метод вычисления перестановки, обратной к ней. Присвоим $Y[X[k]] \leftarrow k$ для $1 \le k \le n$. Тогда $Y[1]Y[2]\ldots Y[n]$ — искомая обратная перестановка. В этом методе используется $2n$ ячеек памяти, а именно — n для X и n для Y.

А теперь ради интереса предположим, что n очень велико, в то время как нужно вычислить обратную перестановку к $X[1]X[2]\ldots X[n]$, не используя слишком много дополнительного пространства памяти. Необходимо вычислить обратную перестановку "на месте", чтобы после окончания работы алгоритма массив $X[1]X[2]\ldots X[n]$ представлял собой перестановку, обратную к первоначальной. Простое присвоение $X[X[k]] \leftarrow k$ для $1 \le k \le n$ в этом случае, разумеется, не пройдет, но, рассматривая циклическую структуру, можно получить следующий простой алгоритм.

Таблица 3

ОБРАЩЕНИЕ ПЕРЕСТАНОВКИ 621543 С ПОМОЩЬЮ АЛГОРИТМА I

После шага:	I2	I3	I3	I3	I5*	I2	I3	I3	I5	I2	I5	I5	I3	I5	I5
$X[1]$	6	6	6	−3	−3	−3	−3	−3	−3	−3	−3	−3	−3	−3	3
$X[2]$	2	2	2	2	2	2	2	2	2	2	2	2	−4	2	2
$X[3]$	1	1	−6	−6	−6	−6	−6	−6	−6	−6	−6	6	6	6	6
$X[4]$	5	5	5	5	5	5	5	−5	−5	−5	5	5	5	5	5
$X[5]$	4	4	4	4	4	4	−1	−1	4	4	4	4	4	4	4
$X[6]$	3	−1	−1	−1	1	1	1	1	1	1	1	1	1	1	1
m	6	3	1	6	6	5	4	5	5	4	4	3	2	2	1
j	−1	−6	−3	−1	−1	−1	−5	−4	−4	−4	−4	−4	−2	−2	−2
i	3	1	6	−1	−1	4	5	−1	−4	−5	−5	−6	−4	−2	−3

Читать столбцы слева направо. В момент * цикл (1 6 3) уже был обращен.

Алгоритм I (*Обратная перестановка на месте*). Заменить $X[1]X[2]\ldots X[n]$, которая является перестановкой чисел $\{1,2,\ldots,n\}$, обратной перестановкой. Этот алгоритм был предложен Бин-Чао Хуаном (Bing-Chao Huang) [*Inf. Proc. Letters* **12** (1981), 237–238].

I1. [Инициализация.] Присвоить $m \leftarrow n$, $j \leftarrow -1$.

I2. [Следующий элемент.] Присвоить $i \leftarrow X[m]$. Если $i < 0$, перейти к шагу I5 (этот элемент уже был обработан).

I3. [Обратить один элемент.] (В этот момент $j < 0$ и $i = X[m]$. Если m не является наибольшим элементом своего цикла, то первоначальная перестановка давала $X[-j] = m$.) Присвоить $X[m] \leftarrow j$, $j \leftarrow -m$, $m \leftarrow i$, $i \leftarrow X[m]$.

I4. [Конец цикла?] Если $i > 0$, перейти к шагу I3 (этот цикл не закончен); иначе — присвоить $i \leftarrow j$. (В последнем случае первоначальная перестановка давала $X[-j] = m$, где m — наибольший элемент в его цикле.)

I5. [Сохранить окончательное значение.] Присвоить $X[m] \leftarrow -i$. (Первоначально $X[-i]$ было равно m.)

I6. [Цикл по m.] Уменьшить m на 1. Если $m > 0$, перейти к I2; в противном случае работа алгоритма заканчивается. ∎

Пример данного алгоритма приведен в табл. 3. Этот метод основан на обращении последовательных циклов перестановки; для пометки обращенных элементов их делают отрицательными, а впоследствии восстанавливают первоначальный знак.

Алгоритм I частично напоминает алгоритм A и очень сильно напоминает алгоритм нахождения циклов, реализованный в программе B (строки 54–68). Таким образом, это типичный представитель алгоритмов, имеющих отношение к перестановкам. Во время подготовки его реализации для MIX было обнаружено, что удобнее всего сохранять в регистре величину $-i$, а не саму i.

Программа I (*Обращение на месте*). rI1 $\equiv m$; rI2 $\equiv -i$; rI3 $\equiv j$ и $n =$ N (этот символ определяется, когда программа транслируется как часть большей программы).

```
01  INVERT ENT1 N      1  I1. Инициализация. m ← n.
02          ENT3 -1    1  j ← −1.
```

```
03  2H      LD2N X,1    N   I2. Следующий элемент. i ← X[m].
04          J2P  5F     N   Переход к шагу I5, если i < 0.
05  3H      ST3  X,1    N   I3. Обратить элемент. X[m] ← j.
06          ENN3 0,1    N   j ← −m.
07          ENN1 0,2    N   m ← i.
08          LD2N X,1    N   i ← X[m].
09  4H      J2N  3B     N   I4. Конец цикла? Переход к шагу I3, если i > 0.
10          ENN2 0,3    C   В противном случае присвоить i ← j.
11  5H      ST2  X,1    N   I5. Сохранить окончательное значение. X[m] ← −i.
12  6H      DEC1 1      N   I6. Цикл по m.
13          J1P  2B     N   Переход к шагу I2, если m > 0. ▮
```

Время выполнения этой программы легко подсчитать способом, о котором говорилось выше. Каждому элементу $X[m]$ сначала присваивается отрицательное значение на шаге I3, а затем — положительное на шаге I5. Общее время выполнения составляет $(14N+C+2)u$, где N — размерность массива, а C — общее число циклов. Ниже будет проанализировано поведение C в случайной перестановке.

Почти всегда существует несколько алгоритмов решения любой поставленной задачи, поэтому можно ожидать, что есть еще один способ обращения перестановки. Следующий остроумный алгоритм был предложен Дж. Бутройдом (J. Boothroyd).

Алгоритм J (*Обращение на месте*). Этот алгоритм дает такой же результат, как и алгоритм I, но на основании другого метода.

J1. [Сделать все величины отрицательными.] Присвоить $X[k] \leftarrow -X[k]$ для $1 \le k \le n$. Присвоить также $m \leftarrow n$.

J2. [Инициализация j.] Присвоить $j \leftarrow m$.

J3. [Нахождение отрицательного элемента.] Присвоить $i \leftarrow X[j]$. Если $i > 0$, присвоить $j \leftarrow i$ и повторить этот шаг.

J4. [Обращение.] Присвоить $X[j] \leftarrow X[-i]$, $X[-i] \leftarrow m$.

J5. [Цикл по m.] Уменьшить m на 1; если $m > 0$, вернуться к шагу J2. В противном случае работа алгоритма заканчивается. ▮

Пример выполнения алгоритма Бутройда приведен в табл. 4. В сущности, в основе метода опять лежит циклическая структура, но в данном случае то, что алгоритм действительно решает поставленную задачу, гораздо менее очевидно. Мы предоставляем читателю проверить это самостоятельно (см. упр. 13).

Программа J (*Аналог программы I*). rI1 ≡ m; rI2 ≡ j; rI3 ≡ $-i$.

```
01  INVERT ENN1 N               1   J1. Сделать все величины отрицательными.
02         ST1  X+N+1,1(0:0)    N   Установить отрицательный знак.
03         INC1 1               N
04         J1N  *-2             N   Еще?
05         ENT1 N               1   m ← n.
06  2H     ENN3 0,1             N   J2. Инициализация j. i ← m.
07         ENN2 0,3             A   j ← i.
08         LD3N X,2             A   J3. Нахождение отрицательного элемента.
```

Таблица 4

ОБРАЩЕНИЕ ПЕРЕСТАНОВКИ 621543 С ПОМОЩЬЮ АЛГОРИТМА J

После шага:	J2	J3	J5	J3	J5	J3	J5	J3	J5	J3	J5	J3	J5
$X[1]$	−6	−6	−6	−6	−6	−6	−6	−6	3	3	3	3	3
$X[2]$	−2	−2	−2	−2	−2	−2	−2	−2	−2	−2	2	2	2
$X[3]$	−1	−1	6	6	6	6	6	6	6	6	6	6	6
$X[4]$	−5	−5	−5	−5	5	5	5	5	5	5	5	5	5
$X[5]$	−4	−4	−4	−4	−5	−5	4	4	4	4	4	4	4
$X[6]$	−3	−3	−1	−1	−1	−1	−1	−1	−6	−6	−6	−6	1
m	6	6	5	5	4	4	3	3	2	2	1	1	0
i		−3	−3	−4	−4	−5	−5	−1	−1	−2	−2	−6	−6
j	6	6	6	5	5	5	5	6	6	2	2	6	6

```
09      J3N   *-2        A   i > 0?
10      LDA   X,3        N   J4. Обращение.
11      STA   X,2        N   X[j] ← X[−i].
12      ST1   X,3        N   X[−i] ← m.
13      DEC1  1          N   J5. Цикл по m.
14      J1P   2B         N   Переход к шагу J2, если m > 0   ▮
```

Чтобы выяснить, насколько быстро работает данная программа, нужно знать величину A; она настолько интересна и поучительна, что мы оставили ее для упражнения (см. упр. 14).

Хотя алгоритм J невероятно изящен, анализ показывает, что алгоритм I намного его превосходит. На самом деле оказывается, что среднее время выполнения алгоритма J, в сущности, пропорционально $n \ln n$, а среднее время выполнения алгоритма I пропорционально n. Возможно, кто-нибудь когда-нибудь найдет применение алгоритму J (или некоторой его модификации); это слишком изящный алгоритм, чтобы его можно было совсем забыть.

Замечательное соответствие. Как уже отмечалось, запись перестановки в циклическом виде не единственна; состоящую из шести элементов перестановку (1 6 3)(4 5) можно записать как (5 4)(3 1 6) и т. д. Поэтому полезно рассмотреть *каноническую форму* циклического представления, которая *является* единственной. Для получения канонической формы выполните следующие действия.

a) Выпишите явно все единичные циклы.
b) Внутри каждого цикла поместите на первое место наименьшее число.
c) Расположите циклы в порядке *убывания* их первых элементов.

Например, для перестановки (3 1 6)(5 4) получим

$$\text{(a): (3 1 6) (5 4) (2);} \quad \text{(b): (1 6 3) (4 5) (2);} \quad \text{(c): (4 5) (2) (1 6 3).} \tag{20}$$

Важным свойством этой канонической формы является то, что скобки можно удалить, а затем восстановить единственным образом. Следовательно, расставить скобки в "4 5 2 1 6 3" для получения канонической циклической формы можно только единственным способом. Необходимо вставить левую скобку непосредственно перед каждым *минимумом слева направо* (т. е. перед тем элементом, перед которым нет меньшего элемента).

Это свойство канонической формы позволяет получить замечательное однозначное соответствие между множеством всех перестановок, представленных в циклическом виде, и множеством всех перестановок, представленных в линейной форме. Например, канонической формой для перестановки 6 2 1 5 4 3 является (4 5) (2) (1 6 3); удалив скобки, получим перестановку 4 5 2 1 6 3, циклической формой которой является (2 5 6 3) (1 4); удалив скобки, получим 2 5 6 3 1 4, циклической формой которой является (3 6 4) (1 2 5) и т. д.

Это соответствие имеет многочисленные применения в изучении перестановок различных типов. Например, зададимся вопросом: "Сколько циклов имеет перестановка из n элементов в среднем?". Чтобы ответить на него, рассмотрим множество всех $n!$ перестановок, представленных в канонической форме, и уберем скобки. Останется множество всех $n!$ перестановок, расположенных в некотором порядке. Поэтому первоначальный вопрос будет эквивалентен следующему: "Сколько в среднем минимумов слева направо имеет перестановка из n элементов?". На последний вопрос мы уже ответили в разделе 1.2.10; это была величина $(A + 1)$ из анализа алгоритма 1.2.10M, для которой найдены статистические характеристики

$$\min 1, \quad \text{ave } H_n, \quad \max n, \quad \text{dev } \sqrt{H_n - H_n^{(2)}}. \tag{21}$$

(На самом деле мы искали среднее число максимумов справа налево, но очевидно, что это то же самое, что число минимумов слева направо.) Более того, мы, в сущности, доказали, что перестановка из n объектов имеет k минимумов слева направо с вероятностью $\left[{n \atop k}\right]/n!$; следовательно, *перестановка из n объектов имеет k циклов с вероятностью* $\left[{n \atop k}\right]/n!$.

Можно также задать вопрос о среднем расстоянии *между* минимумами слева направо, которое эквивалентно средней длине цикла. Согласно (21) *общее* число циклов во всех $n!$ перестановках равно $n!\,H_n$, так как это $n!$, умноженное на *среднее* число циклов. Если выбрать один из этих циклов наугад, то чему будет равна его средняя длина?

Представьте себе все $n!$ перестановок $\{1, 2, \ldots, n\}$, записанных в циклической форме. Сколько среди них будет трехэлементных циклов? Чтобы ответить на этот вопрос, выясним, сколько раз появляется конкретный трехэлементный цикл $(x\,y\,z)$. Очевидно, что он появляется ровно в $(n - 3)!$ перестановках, так как это число способов, которыми можно переставить оставшиеся $n - 3$ элемента. Количество различных возможных трехэлементных циклов $(x\,y\,z)$ равно $n(n-1)(n-2)/3$, поскольку x можно выбрать n способами, y — $(n - 1)$ способами, а z — $(n - 2)$ способами, и среди этих $n(n - 1)(n - 2)$ вариантов каждый отличный от других трехэлементный цикл появляется в трех формах: $(x\,y\,z)$, $(y\,z\,x)$, $(z\,x\,y)$. Поэтому общее число трехэлементных циклов среди всех $n!$ перестановок равно $n(n-1)\times(n-2)/3$, умноженному на $(n-3)!$, т. е. $n!/3$. Аналогично общее число m-элементных циклов равно $n!/m$, где $1 \le m \le n$. (Это дает еще одно простое доказательство того факта, что общее число циклов равно $n!\,H_n$. Следовательно, как мы уже знаем, среднее число циклов в случайной перестановке равно H_n.) В упр. 17 показано, что средняя длина случайно выбранного цикла равна n/H_n, если считать, что все $n!\,H_n$ *циклов* являются равновероятными. Но если *элемент* в случайной перестановке выбран случайно, то средняя длина содержащего его цикла будет несколько больше, чем n/H_n.

Чтобы закончить анализ алгоритмов A и B, следует узнать среднее число *единичных циклов* в случайной перестановке. Это очень интересный вопрос. Предположим, мы записали $n!$ перестановок, перечислив сначала те, в которых нет единичных циклов, затем те, в которых есть только один единичный цикл и т. д. Например, для $n = 4$ это будет выглядеть так.

Нет фиксированных элементов:	2143 2341 2413 3142 3412 3421 4123 4312 4321
Один фиксированный элемент:	$\overline{1}342$ $\overline{1}423$ $3\overline{2}41$ $4\overline{2}13$ $24\overline{3}1$ $41\overline{3}2$ $231\overline{4}$ $312\overline{4}$
Два фиксированных элемента:	$\overline{12}43$ $\overline{14}\overline{3}2$ $\overline{1}32\overline{4}$ $4\overline{23}1$ $3\overline{2}1\overline{4}$ $21\overline{34}$
Три фиксированных элемента:	
Четыре фиксированных элемента:	$\overline{1234}$

(В этом списке отмечены единичные циклы, т. е. элементы, которые в результате перестановки остаются на месте (фиксированные элементы).) Перестановки, не имеющие фиксированных элементов, называются *беспорядочными*; число таких перестановок — это количество способов так разложить n писем по n конвертам, чтобы ни одно из них не попало в свой конверт.

Пусть P_{nk} — число перестановок из n объектов, имеющих ровно k фиксированных элементов. Тогда, например,

$$P_{40} = 9, \quad P_{41} = 8, \quad P_{42} = 6, \quad P_{43} = 0, \quad P_{44} = 1.$$

При изучении приведенного выше списка обнаруживаются главные соотношения, связывающие эти числа. Можно получить все перестановки с k фиксированными элементами, сначала выбирая те k элементов, которые нужно зафиксировать (это можно сделать $\binom{n}{k}$ способами), а затем переставляя оставшиеся $n - k$ элементов всеми $P_{(n-k)0}$ способами, которые больше не оставляют фиксированных элементов. Отсюда

$$P_{nk} = \binom{n}{k} P_{(n-k)0}. \tag{22}$$

Существует также правило, которое говорит о том, что "целое — это сумма составляющих его частей":

$$n! = P_{nn} + P_{n(n-1)} + P_{n(n-2)} + P_{n(n-3)} + \cdots. \tag{23}$$

Подставив (22) в (23) и выполнив простые преобразования, получаем

$$n! = \frac{P_{00}}{0!} + n\frac{P_{10}}{1!} + n(n-1)\frac{P_{20}}{2!} + n(n-1)(n-2)\frac{P_{30}}{3!} + \cdots. \tag{24}$$

Эта формула должна быть справедлива для всех целых положительных n. Она уже встречалась ранее, в разделе 1.2.5 (в связи с попытками Стирлинга обобщить факториальную функцию), а простой вывод ее коэффициентов был приведен в разделе 1.2.6 (пример 5). Мы приходим к выводу, что

$$\frac{P_{m0}}{m!} = 1 - \frac{1}{1!} + \frac{1}{2!} - \cdots + (-1)^m \frac{1}{m!}. \tag{25}$$

Теперь пусть p_{nk} — вероятность того, что перестановка из n объектов имеет ровно k единичных циклов. Так как $p_{nk} = P_{nk}/n!$, из (22) и (25) получаем

$$p_{nk} = \frac{1}{k!}\left(1 - \frac{1}{1!} + \frac{1}{2!} - \cdots + (-1)^{n-k}\frac{1}{(n-k)!}\right). \tag{26}$$

Следовательно, производящую функцию $G_n(z) = p_{n0} + p_{n1}z + p_{n2}z^2 + \cdots$ можно представить в виде

$$G_n(z) = 1 + \frac{1}{1!}(z-1) + \cdots + \frac{1}{n!}(z-1)^n = \sum_{0\le j\le n}\frac{1}{j!}(z-1)^j. \tag{27}$$

Из этой формулы следует, что $G_n'(z) = G_{n-1}(z)$. Воспользовавшись методами, которые применялись в разделе 1.2.10, получим следующие статистические характеристики для числа единичных циклов:

$$(\min\ 0, \quad \text{ave}\ 1, \quad \max\ n, \quad \text{dev}\ 1), \qquad \text{если} \quad n \ge 2. \tag{28}$$

Несколько более прямой способ подсчета числа перестановок, не имеющих единичных циклов, следует из *принципа включения и исключения*, который имеет большое значение для многих задач перечисления. Общий принцип включения и исключения можно сформулировать следующим образом. Дано N элементов и M подмножеств этих элементов $S_1, S_2, \ldots, S_M$. Необходимо подсчитать, сколько элементов не попало ни в одно из этих подмножеств. Пусть $|S|$ — число элементов множества S. Тогда искомое число объектов, не принадлежащих ни одному из множеств S_j, равно

$$N - \sum_{1\le j\le M}|S_j| + \sum_{1\le j<k\le M}|S_j\cap S_k| - \sum_{1\le i<j<k\le M}|S_i\cap S_j\cap S_k| + \cdots + (-1)^M|S_1\cap\cdots\cap S_M|. \tag{29}$$

(Таким образом, сначала из общего числа N вычитается количество элементов множеств $S_1, \ldots, S_M$; но это меньше искомой величины, так как мы вычли больше, чем нужно. Поэтому снова добавляем число элементов, которые являются общими для пар множеств, $S_j \cap S_k$, для каждой пары S_j и S_k; но это уже больше искомой величины. Поэтому вычитаем элементы, общие для каждой тройки множеств, и т. д.) Существует несколько способов доказательства этой формулы, и читателю предлагается найти один из них самостоятельно (см. упр. 25).

Чтобы подсчитать число перестановок из n элементов, не имеющих единичных циклов, рассмотрим $N = n!$ перестановок и обозначим через S_j множество перестановок, в которых элемент j образует единичный цикл. Если $1 \le j_1 < j_2 < \cdots < j_k \le n$, то количество элементов в $S_{j_1}\cap S_{j_2}\cap\cdots\cap S_{j_k}$ — это число перестановок, в которых $j_1, \ldots, j_k$ образуют единичные циклы. Очевидно, что оно равно $(n-k)!$. Таким образом, формула (29) принимает вид

$$n! - \binom{n}{1}(n-1)! + \binom{n}{2}(n-2)! - \binom{n}{3}(n-3)! + \cdots + (-1)^n\binom{n}{n}0!,$$

что согласуется с (25).

Принцип включения и исключения был предложен А. де Муавром (A. de Moivre) [см. его работу *Doctrine of Chances* (London, 1718), 61–63; 3rd ed. (1756, переиздано

Chelsea, 1957), 110–112]. Но значение этого принципа не было по достоинству оценено до тех пор, пока В. А. Витворт (W. A. Whitworth) не популяризировал и не развил его в своей знаменитой книге *Choice and Chance* (Cambridge, 1867).

В разделе 5.1 будет продолжено изучение комбинаторных свойств перестановок.

УПРАЖНЕНИЯ

1. [*02*] Рассмотрите преобразование множества $\{0, 1, 2, 3, 4, 5, 6\}$, которое меняет x на $2x \bmod 7$. Покажите, что это преобразование является перестановкой, и представьте ее в циклической форме.

2. [*10*] В тексте раздела показано, как можно присвоить $(a, b, c, d, e, f) \leftarrow (c, d, f, b, e, a)$, выполнив ряд операций замены $(x \leftarrow y)$ и введя одну вспомогательную переменную t. Покажите, как сделать то же самое с помощью ряда операций *взаимного обмена* $(x \leftrightarrow y)$ и без вспомогательных переменных.

3. [*03*] Вычислите произведение $\left(\begin{smallmatrix} a & b & c & d & e & f \\ b & d & c & a & f & e \end{smallmatrix}\right) \times \left(\begin{smallmatrix} a & b & c & d & e & f \\ c & d & f & b & e & a \end{smallmatrix}\right)$ и запишите результат в виде двухстрочной записи. (Ср. с соотношением (4).)

4. [*10*] Выразите $(a\,b\,d)(e\,f)(a\,c\,f)(b\,d)$ в виде произведения непересекающихся циклов.

▶ **5.** [*M10*] В (3) приведено несколько эквивалентных способов представления одной и той же перестановки в циклической форме. Сколько существует таких представлений перестановки, в которых вообще нет единичных циклов?

6. [*M23*] Как изменится время выполнения программы A, если отказаться от предположения, что все пустые слова появляются у правого края ввода?

7. [*10*] Если на вход программы A подается произведение перестановок (6), то чему равны величины X, Y, M, N, U и V из (19)? Каким будет время выполнения программы A без учета ввода-вывода?

▶ **8.** [*23*] Можно ли модифицировать алгоритм B так, чтобы просматривать входные данные слева направо, а не справа налево?

9. [*10*] Программы A и B воспринимают одинаковые входные данные и ответ дают, в сущности, в одинаковой форме. Дают ли обе программы *в точности* один и тот же вывод?

▶ **10.** [*M28*] Рассмотрите характеристики времени выполнения программы B, а именно — величины A, B, ..., Z, о которых шла речь в тексте раздела. Выразите общее время выполнения через величины X, Y, M, N, U, V, определенные в (19), и через F. Сравните общее время выполнения программ A и B для ввода (6), проведя вычисления, как в упр. 7.

11. [*15*] Найдите простое правило, по которому можно записать π^- в циклической форме, если перестановка π задана в циклической форме.

12. [*M27*] (*Транспонирование прямоугольной матрицы.*) Пусть матрица (a_{ij}) размера $m \times n$, $m \neq n$, хранится в памяти, как описано в упр. 1.3.2–10, так что значение a_{ij} находится в ячейке $L + n(i-1) + (j-1)$, где L — это ячейка, в которой содержится a_{11}. Необходимо найти способ *транспонирования* этой матрицы, чтобы получить матрицу (b_{ij}) размера $n \times m$, где $b_{ij} = a_{ji}$ хранится в ячейке $L + m(i-1) + (j-1)$. Таким образом, наша матрица должна быть транспонирована "на себя". (a) Покажите, что преобразование транспонирования переводит значение из ячейки $L + x$ в ячейку $L + (mx \bmod N)$ для всех x из промежутка $0 \leq x < N = mn - 1$. (b) Обдумайте методы выполнения такого транспонирования с помощью компьютера.

▶ **13.** [*M24*] Докажите справедливость алгоритма J.

▶ **14.** [*M34*] Найдите среднее значение величины A, которая является одной из составляющих времени выполнения алгоритма J.

15. [*M12*] Существует ли перестановка, для которой каноническая циклическая форма без скобок и линейная форма совпадают?

16. [*M15*] Пусть задана перестановка 1324 в линейной форме. Преобразуйте ее в каноническую циклическую форму, а затем удалите скобки. Повторяйте эту процедуру до тех пор, пока не придете к исходной перестановке. Какие перестановки получатся в ходе выполнения этого процесса?

17. [*M24*] (a) В тексте раздела показано, что среди всех перестановок из n элементов существует всего $n!\,H_n$ циклов. Если эти циклы (включая единичные) записать по отдельности на $n!\,H_n$ листочках бумаги, а затем выбрать наугад один из этих листочков, то чему будет равна средняя длина выбранного таким образом цикла? (b) Запишем $n!$ перестановок на $n!$ листочках бумаги, наугад выберем число k и таким же случайным образом вытянем один из листочков. Чему равна вероятность того, что цикл, содержащий элемент k, является m-циклом? Чему равна средняя длина цикла, содержащего элемент k?

▸ **18.** [*M27*] Чему равна p_{nkm}, вероятность того, что перестановка n объектов имеет ровно k циклов длины m? Какой будет соответствующая производящая функция $G_{nm}(z)$? Чему равно среднее число m-циклов и каким будет среднее квадратичное отклонение? (В тексте раздела рассматривается только случай, когда $m = 1$.)

19. [*HM21*] Пользуясь обозначениями из (25), покажите, что число перестановок P_{n0} в точности равно значению $n!/e$, *округленному до ближайшего целого*, для всех $n \geq 1$.

20. [*M20*] Пусть все единичные циклы выписаны явно. Сколько существует различных способов представления в виде циклов перестановки, имеющей α_1 циклов длины 1, α_2 циклов длины 2, ... (см. упр. 5)?

21. [*M22*] Чему равна вероятность $P(n; \alpha_1, \alpha_2, \dots)$ того, что перестановка n объектов имеет ровно α_1 циклов длины 1, α_2 циклов длины 2 и т. д.?

▸ **22.** [*HM34*] (Следующий подход, предложенный Л. Шеппом (L. Shepp) и С. Ф. Ллойдом (S. P. Lloyd), дает удобный и мощный метод решения задач, имеющих отношение к циклическим представлениям случайных перестановок. Вместо того чтобы считать число объектов n фиксированным, а число перестановок — переменным, будем предполагать, что мы независимо выбираем величины $\alpha_1, \alpha_2, \alpha_3 \dots$, рассмотренные в упр. 20 и 21, в соответствии с некоторым распределением вероятностей. Пусть w — любое действительное число между 0 и 1.

a) Предположим, что выбраны случайные переменные $\alpha_1, \alpha_2, \alpha_3, \dots$ согласно правилу, которое гласит: "Вероятность того, что $\alpha_m = k$, равна $f(w, m, k)$", где $f(w, m, k)$ — некоторая функция. Определите функцию $f(w, m, k)$ так, чтобы выполнялись следующие два условия: (i) $\sum_{k \geq 0} f(w, m, k) = 1$, где $0 < w < 1$ и $m \geq 1$; (ii) вероятность того, что $\alpha_1 + 2\alpha_2 + 3\alpha_3 + \cdots = n$ *и* что $\alpha_1 = k_1$, $\alpha_2 = k_2$, $\alpha_3 = k_3$, ..., равна $(1 - w)w^n P(n; k_1, k_2, k_3, \dots)$, где $P(n; k_1, k_2, k_3, \dots)$ определяется в упр. 21.

b) Очевидно, что перестановка, циклическая структура которой имеет вид $\alpha_1, \alpha_2, \alpha_3, \dots$, переставляет ровно $\alpha_1 + 2\alpha_2 + 3\alpha_3 + \cdots$ объектов. Покажите, что если α_i, $i \geq 1$, выбираются случайно в соответствии с распределением вероятностей из п. (a), то вероятность того, что $\alpha_1 + 2\alpha_2 + 3\alpha_3 + \cdots = n$, равна $(1 - w)w^n$, а вероятность того, что сумма $\alpha_1 + 2\alpha_2 + 3\alpha_3 + \cdots$ *бесконечна*, равна нулю.

c) Пусть $\phi(\alpha_1, \alpha_2, \dots)$ — произвольная функция от бесконечного числа аргументов α_1, $\alpha_2, \dots$. Покажите, что если α_i, $i \geq 1$, выбираются в соответствии с распределением вероятностей из п. (a), то среднее значение ϕ равно $(1 - w) \sum_{n \geq 0} w^n \phi_n$. Здесь ϕ_n обозначает среднее значение ϕ для всех перестановок n объектов, а переменная α_j представляет число j-циклов в перестановке. [Например, если $\phi(\alpha_1, \alpha_2, \dots) = \alpha_1$, то значением ϕ_n будет среднее число единичных циклов в случайной перестановке n объектов. Из (28) следует, что $\phi_n = 1$ для всех n.]

d) Используйте этот метод для нахождения среднего числа циклов *четной* длины в случайной перестановке n объектов.

e) Используйте этот метод для решения упр. 18.

23. [*HM42*] (Голомб (Golomb), Шепп (Shepp) и Ллойд (Lloyd).) Обозначим через l_n среднюю длину *самого длинного* цикла в перестановке n объектов. Покажите, что $l_n \approx \lambda n + \frac{1}{2}\lambda$, где $\lambda \approx 0.62433$ — константа. Докажите, что $\lim_{n\to\infty}(l_n - \lambda n - \frac{1}{2}\lambda) = 0$.

24. [*M41*] Найдите дисперсию величины A, которая является одной из характеристик времени выполнения алгоритма J (см. упр. 14).

25. [*M22*] Докажите соотношение (29).

▸ **26.** [*M24*] Обобщите принцип включения и исключения, чтобы получить формулу для числа элементов, которые имеются ровно в r из подмножеств $S_1, S_2, \ldots, S_M$. (В тексте раздела рассматривается только случай $r = 0$.)

27. [*M20*] Используйте принцип включения и исключения для подсчета количества таких целых чисел n в интервале $0 \le n < am_1m_2\ldots m_t$, которые не делятся ни на одно из $m_1, m_2, \ldots, m_t$. Здесь m_1, m_2, ..., m_t и a — положительные целые числа, такие, что $m_j \perp m_k$, если $j \ne k$.

28. [*M21*] (И. Капланси (I. Kaplansky).) Если "перестановку Иосифа", определенную в упр. 1.3.2–22, выразить в циклической форме, то получим (1 5 3 6 8 2 4)(7), где $n = 8$ и $m = 4$. Покажите, что в общем случае эта перестановка представляет собой произведение $(n\ n{-}1\ \ldots\ 2\ 1)^{m-1}(n\ n{-}1\ \ldots\ 2)^{m-1}\ldots(n\ n{-}1)^{m-1}$.

29. [*M25*] Докажите, что циклическую форму перестановки Иосифа при $m = 2$ можно получить, сначала выразив "двойную" перестановку $\{1, 2, \ldots, 2n\}$, которая переводит j в $(2j) \bmod (2n+1)$, в циклической форме, а затем рассмотрев циклы в обратном порядке и убрав все числа, которые больше n. Например, при $n = 11$ двойная перестановка будет иметь вид (1 2 4 8 16 9 18 13 3 6 12)(5 10 20 17 11 22 21 19 15 7 14), а перестановка Иосифа — (7 11 10 5)(6 3 9 8 4 2 1).

30. [*M24*] Используя упр. 29, покажите, что фиксированными элементами перестановки Иосифа при $m = 2$ будут в точности числа $(2^{d-1} - 1)(2n+1)/(2^d - 1)$ для всех таких положительных d, при которых это выражение дает целые числа.

31. [*HM33*] Обобщая упр. 29 и 30, докажите, что k-м казненным для произвольных m и n будет тот, кто находится в позиции x, где x можно вычислить следующим образом: положить $x \leftarrow km$, а затем присваивать $x \leftarrow \lfloor (m(x-n)-1)/(m-1) \rfloor$ до тех пор, пока $x \le n$. В результате среднее число фиксированных элементов для $1 \le n \le N$ и фиксированного m при $N \to \infty$ будет стремиться к $\sum_{k\ge 1}(m-1)^k/(m^{k+1} - (m-1))^k$. [Так как это значение лежит между $(m-1)/m$ и 1, перестановка Иосифа имеет немного меньше фиксированных элементов, чем случайные перестановки.]

32. [*M25*] (a) Докажите, что для любой перестановки $\pi = \pi_1\pi_2\ldots\pi_{2m+1}$ вида

$$\pi = (2\ 3)^{e_2}(4\ 5)^{e_4}\ldots(2m\ 2m{+}1)^{e_{2m}}(1\ 2)^{e_1}(3\ 4)^{e_3}\ldots(2m{-}1\ 2m)^{e_{2m-1}},$$

где каждое e_k — либо 0, либо 1, имеет место $|\pi_k - k| \le 2$ для $1 \le k \le 2m+1$.

(b) Для любой заданной перестановки ρ элементов $\{1, 2, \ldots, n\}$ постройте перестановку π указанного вида, такую, чтобы произведение $\rho\pi$ давало единственный цикл. Таким образом, каждая перестановка "близка" к циклу.

33. [*M33*] Пусть $m = 2^{2^l}$, а $n = 2^{2l+1}$. Покажите, как построить последовательность перестановок $(\alpha_{j1}, \alpha_{j2}, \ldots, \alpha_{jn}; \beta_{j1}, \beta_{j2}, \ldots, \beta_{jn})$ для $0 \le j < m$, имеющих следующее свойство "ортогональности":

$$\alpha_{i1}\beta_{j1}\alpha_{i2}\beta_{j2}\ldots\alpha_{in}\beta_{jn} = \begin{cases} (1\,2\,3\,4\,5), & \text{если } i = j; \\ (), & \text{если } i \ne j. \end{cases}$$

Каждое α_{jk} и β_{jk} должно быть перестановкой чисел $\{1, 2, 3, 4, 5\}$.

▸ **34.** [*M25*] (*Транспонирующие блоки данных.*) Одной из самых распространенных перестановок, использующихся на практике, является переход от $\alpha\beta$ к $\beta\alpha$, где α и β — это подстроки массива. Другими словами, если $x_0x_1\ldots x_{m-1} = \alpha$ и $x_mx_{m+1}\ldots x_{m+n-1} = \beta$, то нужно заменить массив $x_0x_1\ldots x_{m+n-1} = \alpha\beta$ массивом $x_mx_{m+1}\ldots x_{m+n-1}x_0x_1\ldots x_{m-1} = \beta\alpha$. Это перестановка на множестве $\{0, 1, \ldots, m+n-1\}$, которая переводит k в $(k+m) \bmod (m+n)$. Покажите, что каждая такая перестановка "с циклическим сдвигом" имеет простую циклическую структуру, и используйте эту структуру для разработки простого алгоритма получения нужной перестановки.

35. [*M30*] В продолжение предыдущего упражнения положим $x_0x_1\ldots x_{l+m+n-1} = \alpha\beta\gamma$, где α, β и γ — строки длины l, m и n соответственно, и предположим, что нужно заменить $\alpha\beta\gamma$ на $\gamma\beta\alpha$. Покажите, что соответствующая перестановка имеет подходящую циклическую структуру, которая позволяет получить эффективный алгоритм. [Упр. 34 рассматривалось как частный случай при $m = 0$.] *Указание.* Рассмотрите замену $(\alpha\beta)(\gamma\beta)$ на $(\gamma\beta)(\alpha\beta)$.

36. [*27*] Напишите для `MIX` подпрограмму, реализующую алгоритм, который приведен в ответе к упр. 35, и проанализируйте время его выполнения. Сравните этот алгоритм с более простым методом, в котором осуществляется переход от $\alpha\beta\gamma$ к $(\alpha\beta\gamma)^R = \gamma^R\beta^R\alpha^R$ и к $\gamma\beta\alpha$, где σ^R обозначает полное обращение строки σ, т. е. строка читается в обратном порядке.

1.4. НЕКОТОРЫЕ ФУНДАМЕНТАЛЬНЫЕ МЕТОДЫ ПРОГРАММИРОВАНИЯ

1.4.1. Подпрограммы

КОГДА НЕКОТОРУЮ задачу нужно выполнить в нескольких различных местах программы, то, как правило, нежелательно каждый раз повторять ее код. Чтобы избежать этого, данный код (называемый *подпрограммой*) можно поместить только в одном месте и добавить несколько дополнительных команд, чтобы после завершения работы подпрограммы должным образом возобновить выполнение внешней программы. Передача управления между подпрограммами и основной программой называется *связью с подпрограммами*.

Каждый компьютер имеет свои специфические способы установления эффективной связи с подпрограммами, которые обычно подразумевают применение специальных команд. В MIX для этой цели используется регистр J. Рассматривая данную тему, будем ориентироваться на машинный язык MIX, но аналогичные рассуждения применимы и к вопросам связи с подпрограммами для других компьютеров.

Подпрограммы используются с целью экономии места в программе, но их применение не приводит непосредственно к экономии времени. Это происходит неявно вследствие того, что программа занимает меньше места в памяти, например меньше времени тратится на загрузку программы, уменьшается количество проходов в программе либо более эффективно используется высокоскоростная память на машинах с несколькими уровнями памяти. Дополнительным временем, которое тратится на вход в подпрограмму и выход из нее, обычно можно пренебречь.

Подпрограммы имеют и другие преимущества. Благодаря им структура больших и сложных программ становится более наглядной. Они разбивают всю задачу на логические сегменты, что обычно облегчает отладку программы. Многие подпрограммы имеют дополнительную ценность, поскольку ими могут воспользоваться не только авторы, но и другие пользователи.

Большинство компьютерных средств разработки имеют обширные встроенные библиотеки полезных подпрограмм, что значительно облегчает программирование стандартных прикладных задач. Но программист не должен думать, что подпрограммы предназначены только для какой-то *одной* цели. Не следует всегда рассматривать подпрограммы только как программы общего назначения, которыми все могут пользоваться. Не менее важны и подпрограммы специального назначения, даже если они используются только в одной программе. В разделе 1.4.3.1 рассматриваются типичные примеры подпрограмм.

Простейшими являются подпрограммы, которые имеют только один вход и один выход, как, например, подпрограмма MAXIMUM, рассмотренная выше (см. раздел 1.3.2, программа M). Еще раз приведем текст этой программы, изменив ее таким образом, чтобы поиск максимума велся по фиксированному числу ячеек, равному 100.

```
* MAXIMUM OF X[1..100]
MAX100  STJ  EXIT   Связь с подпрограммой.
        ENT3 100    M1. Инициализация.
        JMP  2F
1H      CMPA X,3    M3. Сравнение.
        JGE  *+3                                   (1)
2H      ENT2 0,3    M4. Замена m.
        LDA  X,3    Найден новый максимум.
        DEC3 1      M5. Уменьшение k.
        J3P  1B     M2. Все проверено?
EXIT    JMP  *      Вернуться к основной программе. ∎
```

В большой программе, содержащей этот код в качестве подпрограммы, с помощью единственной команды "JMP MAX100" можно занести в регистр A значение текущего максимума для ячеек с X + 1 по X + 100 и поместить информацию о положении максимума в rI2. Связь с подпрограммой в этом случае достигается с помощью команд "MAX100 STJ EXIT" и позднее — "EXIT JMP *". Регистр J работает таким образом, что команда выхода затем перейдет в ячейку, следующую за той, из которой было сделано первоначальное обращение к MAX100.

В более новых модификациях компьютеров, таких как машина MMIX, которой суждено заменить MIX, предусмотрены более эффективные способы запоминания адресов возврата. Главное отличие состоит в том, что команды программы больше не модифицируются в памяти; необходимая информация сохраняется в регистрах или в специальном массиве, а не в самой программе (см. упр. 7). В следующем издании данной книги будет применен современный подход к этим вопросам, а пока будем по-прежнему использовать старый самомодифицирующийся код.

Нетрудно получить *количественные* характеристики степени экономичности кода и потери времени при использовании подпрограмм. Предположим, некоторый фрагмент кода занимает k ячеек и встречается в m местах программы. Чтобы оформить этот фрагмент в качестве подпрограммы, понадобится дополнительная команда STJ, строка для команды выхода из подпрограммы плюс по одной команде JMP в каждом из m мест, откуда будет вызываться подпрограмма. В целом, необходимо $m + k + 2$ ячеек, а не mk, поэтому экономия составляет

$$(m - 1)(k - 1) - 3 \tag{2}$$

ячеек. Если k равно 1 либо m равно 1, то, очевидно, мы не сможем сэкономить место в памяти за счет использования подпрограмм. Если k равно 2, то для получения выигрыша m должно быть больше 4, и т. д.

Время теряется из-за применения дополнительных команд JMP, STJ и JMP, которые не присутствуют в программе, если подпрограмма не используется. Поэтому, если во время выполнения основной программы подпрограмма применяется t раз, потребуется $4t$ дополнительных тактов.

К этим оценкам следует подходить с определенной долей скепсиса, так как они делались в расчете на идеальную ситуацию. Многие подпрограммы нельзя вызвать просто с помощью единственной команды JMP. Более того, если фрагмент кода повторяется во многих частях программы и он не оформляется в виде подпрограммы, то для каждой части данный код можно модифицировать так, чтобы получать преимущества от особых характеристик конкретной части программы, в которой он находится. С другой стороны, если принято решение использовать подпрограмму, то ее код нужно писать для наиболее общего, а не частного, случая, и обычно это требует добавления нескольких дополнительных команд.

Если подпрограмма написана для общего случая, то она, как правило, зависит от *параметров*. Параметры — это величины, которые управляют работой подпрограммы; они могут изменяться от одного вызова подпрограммы к другому.

Фрагмент кода внешней программы, который передает управление подпрограмме и должным образом запускает ее, называется *последовательностью вызова*. Конкретные значения параметров, которые передаются при вызове подпрограммы, называются *аргументами*. В нашей подпрограмме MAX100 вызывающая последова-

тельность — это просто команда "JMP MAX100". Но в случае, когда необходимо передать аргументы, обычно необходима более длинная вызывающая последовательность. Например, программа 1.3.2M — это обобщенный вариант MAX100; она находит максимум среди первых n элементов таблицы. Параметр n появляется в индексном регистре 1, и его последовательность вызова

```
LD1  =n=                 ENT1  n
JMP  MAXIMUM     или     JMP   MAXIMUM
```

включает два шага.

Если последовательность вызова занимает c ячеек памяти, то формула (2) вычисления объема сэкономленного места принимает вид

$$(m-1)\,(k-c) - \text{constant}, \tag{3}$$

а время, которое тратится на связь с подпрограммой, немного увеличивается.

Может возникнуть необходимость внесения дальнейших корректив в приведенные формулы, если потребуется сохранить и восстановить некоторые регистры. Например, работая с подпрограммой MAX100, следует помнить, что, записывая "JMP MAX100", мы не только заносим максимальное значение в регистр A, а его позицию — в регистр I2, но и обнуляем регистр I3. Нужно иметь в виду, что подпрограмма может испортить содержимое регистров. Чтобы предотвратить изменение подпрограммой MAX100 содержимого rI3, необходимо включить дополнительные команды. Самый короткий и самый быстрый способ сделать это на MIX состоит в том, чтобы вставить команду "ST3 3F(0:2)" сразу после MAX100 и команду "3H ENT3 *" — непосредственно перед EXIT. В итоге это выльется в две дополнительные строки кода плюс три машинных такта для каждого вызова подпрограммы.

Подпрограмму можно рассматривать как *расширение* машинного языка компьютера. Внеся в память машины подпрограмму MAX100, получим единственную команду (а именно — "JMP MAX100"), которая находит максимум. Важно определить результат работы каждой подпрограммы так же тщательно, как были определены сами операторы машинного языка. Поэтому программист обязательно должен записать характеристики каждой подпрограммы, даже если больше никто не будет пользоваться этой программой или ее спецификацией. Например, подпрограмма MAXIMUM, приведенная в разделе 1.3.2, имеет следующие характеристики.

$$\left.\begin{array}{ll} \text{Последовательность} & \\ \text{вызова:} & \texttt{JMP MAXIMUM}. \\ \text{Состояние при входе:} & \text{rI1} = n; \quad \text{в предположении, что } n \geq 1. \\ \text{Состояние при выходе:} & \text{rA} = \max\limits_{1\leq k\leq n} \texttt{CONTENTS(X}+k\texttt{)} = \texttt{CONTENTS(X}+\text{rI2}\texttt{)}; \\ & \text{rI3} = 0\text{; содержимое rJ и CI также меняется.} \end{array}\right\} \tag{4}$$

(Обычно мы не будем упоминать о том, что подпрограмма оказывает влияние на регистр J и флаг сравнения; здесь об этом говорится только для полноты.) Заметьте, что на rX и rI1 подпрограмма действия не оказывает, в противном случае эти регистры были бы упомянуты в описании состояния при выходе. В спецификации должны также перечисляться все внешние для подпрограммы ячейки памяти, на которые она может оказать воздействие. В нашем случае спецификация позволяет заключить, что в памяти ничего не сохранялось, так как в (4) ничего не говорится об изменениях в памяти.

А теперь давайте рассмотрим *несколько входов* в подпрограммы Предположим, существует программа, которой требуется общая подпрограмма MAXIMUM Но дело в том, что обычно используется частный случай MAX100. для которого $n = 100$. Эти две подпрограммы можно объединить следующим образом.

```
MAX100 ENT3 100    Первый вход
MAXN   STJ  EXIT   Второй вход
       JMP  2F     Продолжать, как в (1)                    (5)
...
EXIT   JMP  *      Вернуться к основной программе ▮
```

Подпрограмма (5) практически такая же, как (1), только первые две команды поменялись местами. Здесь использовался тот факт, что команда "ENT3" не изменяет содержимое регистра J. Чтобы добавить к этой подпрограмме *третий* вход, MAX50, в начало нужно вставить строки

```
MAX50  ENT3 50
       JSJ  MAXN .                                          (6)
```

(Напоминаю, что "JSJ" означает переход без изменения регистра J.)

Если число параметров невелико, то желательно передавать их в подпрограмму одним из двух способов· занеся их в подходящие регистры (аналогично тому. как мы использовали rI3 для хранения параметра n в MAXN, а rI1 — для хранения параметра n в MAXIMUM) либо сохранив их в фиксированных ячейках памяти.

Другой удобный способ передачи аргументов состоит в том, чтобы просто перечислить их *после* команды JMP. Подпрограмма сможет обратиться к своим параметрам, поскольку она знает содержимое регистра J. Например, если бы понадобилось создать для MAXN последовательность вызова

```
       JMP  MAXN
       CON  n  ,                                            (7)
```

то подпрограмму можно было бы написать следующим образом:

```
MAXN   STJ  *+1
       ENT1 *     rI1 ← rJ
       LD3  0,1   rI3 ← n
       JMP  2F    Продолжать, как в (1)                     (8)
...
       J3P  1B
       JMP  1,1   Возврат ▮
```

На таких машинах, как IBM 360, на которых связь с подпрограммами обычно осуществляется путем помещения адреса ячейки возврата в индексный регистр, приведенный выше способ особенно удобен. Он используется также, когда у подпрограммы много аргументов либо если программа создана компилятором Метод нескольких входов, который использовался выше, в этом случае не подходит. Его можно "подделать", написав

```
MAX100 STJ  1F
       JMP  MAXN
       CON  100
1H     JMP  *  ,
```

но это выглядит уже не так привлекательно, как (5).

Для подпрограмм с *несколькими выходами* обычно используется метод, аналогичный перечислению аргументов. Множественный выход означает, что подпрограмма возвращается в одно из нескольких различных мест в зависимости от условий, обнаруженных подпрограммой. В самом строгом смысле место, в которое возвращается подпрограмма после выхода, — это параметр. Поэтому, если существует несколько мест, в которые она может выйти в зависимости от обстоятельств, они должны быть предоставлены в качестве аргументов. В нашем завершающем примере в подпрограмме поиска максимума будет два входа и два выхода. Последовательность вызова имеет такой вид.

Для произвольного n	Для $n = 100$
`ENT3` n	
`JMP MAXN`	`JMP MAX100`
Выйти здесь,	Выйти здесь,
если max ≤ 0 или max $\ge$ rX.	если max ≤ 0 или max $\ge$ rX.
Выйти здесь, если $0 <$ max $<$ rX.	Выйти здесь, если $0 <$ max $<$ rX.

(Другими словами, выход производится на *две* ячейки ниже команды перехода, если максимум положителен и меньше значения, которое содержится в регистре X.) Для этих условий подпрограмму написать несложно.

```
MAX100 ENT3 100    Вход для n = 100
MAXN   STJ  EXIT   Вход для произвольного n
       JMP  2F     Далее, как в (1)
...
       J3P  1B
       JANP EXIT   Выполнить нормальный выход, если max ≤ 0          (9)
       STX  TEMP
       CMPA TEMP
       JGE  EXIT   Выполнить нормальный выход, если max ≥ rX.
       ENT3 1      В противном случае выйти через второй выход
EXIT   JMP  *,3    Вернуться в нужное место  ▮
```

Одни подпрограммы могут вызывать другие подпрограммы. В сложных программах вызовы подпрограмм, вложенных более чем на пять уровней, — это не такое уж редкое явление. Используя описанную выше связь подпрограмм, необходимо придерживаться единственного ограничения — одна подпрограмма не может вызывать другую подпрограмму, которая (прямо или косвенно) вызывает ее. Например, рассмотрим следующий сценарий.

```
[Главная программа]   [Подпрограмма A]   [Подпрограмма B]   [Подпрограмма C]
                      A     STJ EXITA     B     STJ EXITB     C     STJ EXITC
        ⋮                     ⋮                   ⋮                   ⋮
      JMP A                 JMP B               JMP C               JMP A
        ⋮                     ⋮                   ⋮                   ⋮
                      EXITA JMP *         EXITB JMP *         EXITC JMP *   (10)
```

Если главная программа вызывает A, которая вызывает B, которая вызывает C, а затем C вызывает A, то адрес в ячейке EXITA, по которому осуществляется выход

в главную программу, затирается, что делает возврат к этой программе невозможным. Подобные рассуждения применимы ко всем ячейкам оперативной памяти и к регистрам, используемым подпрограммами. Поэтому разработать правила связи подпрограмм, которые позволят правильно обрабатывать такие рекурсивные ситуации, совсем несложно; подробности приводятся в главе 8.

В завершение этого раздела кратко рассмотрим подходы к написанию сложных и больших программ. Как узнать, какие подпрограммы нам нужны и какие последовательности вызова нужно использовать? Чтобы успешно решить эту задачу, можно воспользоваться методом итераций.

Шаг 0 (Первоначальная идея). Сначала приблизительно выбираем генеральный план действий, которые будут выполняться в программе.

Шаг 1 (Черновая схема программы). Начнем с написания "внешних уровней" программы на любом удобном языке. Систематизированный подход к этому этапу очень хорошо описан в книге E. W. Dijkstra, *Structured Programming* (Academic Press, 1972), Chapter 1, и в работе N. Wirth, *CACM* **14** (1971), 221–227. Начать можно с разбиения всей программы на небольшие фрагменты, которые временно можно рассматривать как подпрограммы, хотя они вызываются только один раз. Эти фрагменты можно последовательно разбивать на все более мелкие части, которые будут служить для выполнения все более простых операций. Каждый раз, когда возникает какая-либо вычислительная задача, которая, похоже, встретится где-то еще либо уже где-то встретилась, следует определить подпрограмму (уже не гипотетическую, а реальную) для выполнения этой задачи. На данном этапе еще не следует писать эту подпрограмму; нужно продолжить написание главной программы, исходя из предположения, что подпрограмма выполнила свою задачу. И наконец, когда схема главной программы будет готова, можно приниматься за подпрограммы, стараясь начинать с самых сложных подпрограмм, постепенно переходя к вложенным в них подпрограммам и т. д. В результате получится список подпрограмм. Функция каждой подпрограммы, вероятно, менялась уже несколько раз, так что к этому моменту начальные элементы схемы будут неправильными. Ничего страшного, ведь это всего лишь схема. Зато теперь мы имеем достаточно четкое представление о том, как будет вызываться каждая подпрограмма и какова степень ее универсальности. Как правило, имеет смысл сделать каждую подпрограмму более универсальной.

Шаг 2 (Первая рабочая программа). Этот шаг по сравнению с шагом 1 — движение в противоположном направлении. Теперь будем использовать компьютерный язык программирования, скажем, `MIXAL`, `PL/MIX` или язык высокого уровня. На этот раз начнем с самых низких уровней вложения подпрограмм, затем "поднимемся вверх" и в конце концов напишем главную программу. Рекомендуется, по возможности, не писать какие-либо команды вызова подпрограммы до того, как будет написан код самой подпрограммы. (На шаге 1 мы старались делать прямо противоположное, не рассматривая подпрограмму до тех пор, пока не будут написаны все ее последовательности вызова.)

По мере написания все большего и большего числа подпрограмм наша уверенность будет постепенно расти, так как мы постоянно расширяем возможности программируемого модуля. После написания отдельной подпрограммы необходимо сразу же подготовить полное описание ее функций и ее последовательностей вызова,

как в (4). Важно также, чтобы не перекрывались ячейки временной памяти. Если каждая подпрограмма будет обращаться к ячейке TEMP, то последствия могут быть катастрофическими, но во время разработки схемы на этапе 1 нам было удобно не беспокоиться об этом. Очевидный способ решения проблем с перекрытием состоит в следующем: нужно выделить для каждой подпрограммы отдельный участок временной памяти, чтобы его использовала только она. Но если такое использование пространства памяти является слишком расточительным, то можно использовать другую схему — присвоить ячейкам имена TEMP1, TEMP2 и т. д. Нумерация внутри подпрограммы начинается с TEMPj, где j на единицу больше, чем наибольший номер, который был использован какой-либо вложенной подпрограммой этой подпрограммы.

Шаг 3 (Повторная проверка). Результатом шага 2 должна быть практически рабочая программа, но вполне возможно, что ее удастся улучшить. Для этого существует хороший метод — снова сменить направление, исследуя для каждой подпрограммы *все* сделанные по отношению к ней вызовы. Может оказаться, что подпрограмму необходимо несколько расширить, чтобы она выполняла распространенные операции, которые всегда осуществляет внешняя программа непосредственно перед использованием подпрограммы или после него. Возможно, несколько подпрограмм следует объединить в одну; а может быть, некоторая подпрограмма вызывается только один раз и из нее вообще не стоит делать подпрограмму. (Случается и такое: подпрограмма не вызывается ни разу и можно вообще обойтись без нее.)

На этом этапе очень часто имеет смысл выбросить все, что было сделано, и снова начать с шага 1! Я вовсе не шучу; время, потраченное на то, чтобы добраться до этого места, не пропало даром, так как вы достаточно глубоко изучили поставленную задачу. Впоследствии (уже после запуска программы), скорее всего, станет ясно, что в общую схему программы необходимо было внести некоторые улучшения. Поэтому нет причин бояться возврата к шагу 1 — намного легче снова пройти шаги 2 и 3 сейчас, когда аналогичная программа уже готова. Скажу еще больше: вполне возможно, что время, которое будет затрачено на переписывание всей программы, с лихвой окупится впоследствии при отладке. Некоторые лучшие из когда-либо написанных компьютерных программ своим успехом во многом обязаны тому, что примерно на этом этапе вся работа была случайно потеряна и авторам пришлось все начать сначала.

С другой стороны, вероятно, в принципе не может наступить момент, когда сложную компьютерную программу нельзя каким-либо образом улучшить. Поэтому шаги 1 и 2 не следует повторять до бесконечности. Когда совершенно очевидно, что можно внести значительное улучшение, имеет смысл потратить дополнительное время на то, чтобы начать все сначала. Но в конце концов наступает стадия "насыщения", когда сколько-нибудь существенные изменения внести уже нельзя и результатом их воплощения является лишь незначительный прогресс.

Шаг 4 (Отладка). После окончательной "шлифовки" программы, к которой, видимо, относится распределение памяти и другие последние приготовления, самое время посмотреть на нее под еще одним углом зрения, отличным от тех, которые использовались на шагах 1, 2 и 3. Теперь мы будем исследовать элементы программы в том порядке, в котором их будет *выполнять* компьютер. Это можно сделать вручную или, конечно, с помощью компьютера. Автор пришел к выводу,

что на данном этапе очень полезно использовать системные программы, которые прослеживают каждую команду, когда она выполняется первые два раза. Очень важно заново продумать идеи, лежащие в основе программы, и убедиться в том, что все действительно происходит так, как ожидалось.

Отладка — это искусство, которое требует дальнейшего изучения, и способ ее проведения в значительной степени зависит от имеющихся на компьютере устройств. Хорошим началом эффективной отладки во многих случаях может стать подготовка соответствующих тестовых данных. Самыми эффективными, похоже, являются те методы отладки, которые предназначены для конкретной программы и встроены именно в нее. Многие из лучших программистов современности почти половину своих программ посвятили тому, чтобы облегчить отладку программ из другой половины. "Первая половина", которая обычно состоит из достаточно простых программ, отображающих соответствующую информацию в удобном для чтения виде, в конце концов выбрасывается, но в итоге это дает удивительный выигрыш в производительности.

Еще один хороший метод отладки состоит в том, чтобы вести учет всех сделанных ошибок. Конечно, это нелегко, но подобная информация является бесценной для каждого, кто пытается отладить программу; она также поможет в дальнейшем избежать множества ошибок.

Замечание. Большинство предыдущих комментариев было написано автором в 1964 году, после того как он успешно завершил несколько программных проектов среднего масштаба, но до того как он приобрел зрелый стиль программирования. Впоследствии, в 80-х годах, он понял, что, вероятно, еще более важным является метод, который называется *структурным документированием* или *грамотным программированием*. Анализ современных представлений об оптимальных способах написания всевозможных программ содержится в книге *Literate Programming* (Cambridge Univ. Press), впервые опубликованной в 1992 году. Кстати, в главе 11 этой книги приведен подробный перечень всех ошибок, которые были устранены из программы TEX в период между 1978 и 1991 годами.

До некоторого момента лучше допустить наличие ошибок в программе, чем потратить на ее разработку столько времени, сколько необходимо для устранения всех ошибок (сколько десятилетий на это потребуется?).

— А. М. ТЬЮРИНГ, *Proposals for ACE* (1945)

УПРАЖНЕНИЯ

1. [*10*] Сформулируйте характеристики подпрограммы (5). В качестве образца используйте (4), где даются характеристики подпрограммы 1.3.2M.

2. [*10*] Предложите код, заменяющий (6). Не используйте команду JSJ.

3. [*M15*] Дополните информацию, которая дана в (4), точно указав, что происходит с регистром J и флагом сравнения при выполнении подпрограммы. Определите также, что происходит в случае, когда в регистре I1 содержится неположительное число.

▶ **4.** [*21*] Напишите обобщающую MAXN подпрограмму нахождения максимального значения для последовательности X[a], X[$a+r$], X[$a+2r$], ..., X[n], где r и n — параметры, а a — наименьшее положительное число, для которого $a \equiv n$ (по модулю r), т. е.

$a = 1 + (n-1) \bmod r$. Предусмотрите специальный вход для случая $r = 1$. Перечислите характеристики новой подпрограммы, взяв за образец (4).

5. [*21*] Предположим, в машине MIX нет регистра J. Придумайте способ связи подпрограмм, не использующий регистр J. Проиллюстрируйте свое изобретение на примере, написав подпрограмму MAX100, фактически эквивалентную (1). Сформулируйте характеристики этой подпрограммы аналогично тому, как это сделано в (4). (Придерживайтесь принятых для MIX соглашений о самомодифицирующемся коде.)

▸ **6.** [*26*] Предположим, в MIX нет оператора MOVE. Напишите подпрограмму с именем MOVE, такую, чтобы ее последовательность вызова "JMP MOVE; NOP A,I(F)" давала такой же эффект, как и "MOVE A,I(F)", если бы последнее было допустимо. Единственные отличия должны заключаться в воздействии на регистр J и в том, что время выполнения подпрограммы несколько увеличится.

▸ **7.** [*20*] Почему к самомодифицирующемуся коду сейчас относятся неодобрительно?

1.4.2. Сопрограммы

Подпрограммы — это частные случаи более общих программных компонентов, называемых *сопрограммами*. В противоположность несимметричной связи между главной программой и подпрограммой, между сопрограммами, которые *вызывают одна другую*, существует полная симметрия.

Чтобы понять, что представляет собой сопрограмма, давайте посмотрим на подпрограммы с другой стороны. В предыдущем разделе мы придерживались той точки зрения, что подпрограмма — это некая аппаратная реализация, которая используется для сокращения количества строк в программе. Это, конечно, правильно, но возможна и другая точка зрения. Можно рассматривать главную программу и подпрограмму как *группу* программ, причем каждый член группы должен выполнять определенную работу. Главная программа в процессе своей работы активизирует подпрограмму, последняя выполняет собственную функцию, а затем активизирует главную программу. Мы можем выйти за узкие рамки своих представлений и вообразить, что с точки зрения подпрограммы, когда она осуществляет выход, именно *она* вызывает *главную* программу. Главная программа продолжает выполнять свои обязанности, а затем "выходит" в подпрограмму. Подпрограмма работает, а затем снова вызывает главную программу.

Эта несколько надуманная философия на самом деле полностью отражает ситуацию с сопрограммами, для которых невозможно определить, где главная программа, а где подпрограмма. Предположим, имеются сопрограммы A и B. Программируя A, можно считать, что B — это подпрограмма; в свою очередь, программируя B, как подпрограмму можно рассматривать уже A. Таким образом, в сопрограмме A команда "JMP B" используется для активизации сопрограммы B. В сопрограмме B команда "JMP A" применяется для того, чтобы снова активизировать сопрограмму A. Каждый раз при активизации сопрограмма возобновляет выполнение своей программы с той точки, в которой действие было приостановлено в прошлый раз.

Сопрограммами A и B могут быть, например, две программы, играющие в шахматы. Их можно скомбинировать так, чтобы они играли одна против другой.

В MIX подобная связь между сопрограммами A и B осуществляется путем включения в программу следующих четырех команд.

```
A   STJ  BX          B   STJ  AX
AX  JMP  A1          BX  JMP  B1
```
(1)

Для передачи управления любой из сопрограмм требуется четыре машинных такта. Первоначально AX и BX установлены на переход к начальным точкам каждой сопрограммы, A1 и B1. Предположим, сначала запускается сопрограмма A; она начинает выполняться с команды, которая находится в ячейке A1. Когда сопрограмма A выполняет, скажем, команду "JMP B" из ячейки A2, команда из ячейки B сохраняет в AX содержимое rJ, скажем, команду "JMP A2+1". Команда из BX переносит нас в ячейку B1, и вслед за этим начинает выполняться сопрограмма B. В конце концов она доходит до команды "JMP A", которая находится, скажем, в ячейке B2. Мы сохраняем содержимое rJ в BX и переходим к ячейке A2+1, продолжая выполнять сопрограмму A до тех пор, пока управление снова не перейдет к B, которая сохраняет содержимое регистра J в AX, переходит к B2+1 и т. д.

Главное различие между связями "программа — подпрограмма" и "сопрограмма — сопрограмма", как видно из предыдущего примера, состоит в том, что подпрограмма всегда начинается *с самого начала* (как правило, это фиксированная точка), а главная программа или сопрограмма всегда начинается *с места, следующего* за той точкой, в которой ее выполнение было прекращено в прошлый раз.

Необходимость использования сопрограмм на практике естественным образом возникает в случае, когда они связаны с алгоритмами ввода и вывода. Приведем такой пример. Предположим, что в обязанности сопрограммы A входит считывание перфокарт и выполнение такого преобразования входных данных, которое сводит их к последовательности элементов. Другая сопрограмма, которую мы будем называть B, выполняет дальнейшую обработку этих элементов и печатает ответы. B периодически запрашивает последующие элементы ввода, полученные A. Таким образом, сопрограмма B вызывает A каждый раз, когда ей нужен следующий элемент ввода, и сопрограмма A вызывает B каждый раз, когда находит элемент ввода. Читатель может сказать: "Ну что ж, B — это главная программа, а A — просто *подпрограмма* для выполнения ввода". Но это утверждение становится уже не таким правильным, когда процесс A оказывается очень сложным. В действительности можно считать, что A — это главная программа, а B — подпрограмма, выполняющая вывод, и приведенное выше описание останется верным. Но полезность идеи сопрограммы обнаруживается между этими двумя крайностями, когда и A, и B достаточно сложны и каждая из них вызывает другую много раз из различных точек. Подобрать небольшие, простые примеры сопрограмм, иллюстрирующие важность этой идеи, весьма трудно. В случаях, когда применение сопрограмм приносит наибольшую пользу, эти сопрограммы, как правило, оказываются довольно большими.

Чтобы рассмотреть сопрограммы в действии, воспользуемся таким примером. Предположим, необходимо написать программу, преобразующую один код в другой. Входной код, который нужно преобразовать, представляет собой последовательность буквенно-цифровых символов, заканчивающуюся точкой, например

```
A2B5E3426FG0ZYW3210PQ89R.
```
(2)

Эта последовательность выбивается на перфокартах; пустые колонки, встречающиеся на перфокартах, игнорируются. Входные данные интерпретируются следующим образом, слева направо: если следующий символ — цифра 0, 1, ..., 9 (обозначим ее через n), то он указывает на $(n+1)$ повторение следующего символа независимо от того, является ли он цифрой. Нецифровой символ обозначает сам себя. Выходные данные нашей программы должны представлять собой последовательность, полученную указанным образом и разделенную на группы по три символа. Последовательность заканчивается с появлением точки; в последней группе может быть менее трех символов. Например, последовательность (2) должна быть преобразована нашей программой в следующие группы символов:

```
ABB BEE EEE E44 446 66F GZY W22 220 OPQ 999 999 999 R.        (3)
```

Обратите внимание, что 3426F означает не 3427 повторений буквы F, а четыре четверки и три шестерки, за которыми следует буква F. Если входная последовательность имеет вид '1.', то на выходе будет просто '.', а не '..', поскольку первая же точка заканчивает вывод. Наша программа должна выбить выходные данные на перфокартах — по шестнадцать групп на каждой карте, за исключением, возможно, последней

Для выполнения такого преобразования напишем две сопрограммы и одну подпрограмму. Подпрограмма с именем NEXTCHAR предназначена для нахождения непустых символов (т. е не пробелов) во входных данных и помещения каждого последующего такого символа в регистр A.

```
01  * ПОДПРОГРАММА ВВОДА СИМВОЛОВ
02  READER   EQU  16              Номер устройства чтения перфокарт
03  INPUT    ORIG *+16            Место для входных карт
04  NEXTCHAR STJ  9F              Вход в подпрограмму
05           JXNZ 3F              Первоначально rX = 0
06  1H       J6N  2F              Первоначально rI6 = 0
07           IN   INPUT(READER)   Читать следующую карту
08           JBUS *(READER)       Ожидать завершения
09           ENN6 16              Пусть rI6 указывает на следующее слово.
10  2H       LDX  INPUT+16,6      Взять следующее слово из ввода.
11           INC6 1               Продвинуть указатель
12  3H       ENTA 0
13           SLAX 1               Следующий символ → rA
14  9H       JANZ *               Пропустить пробелы
15           JMP  NEXTCHAR+1      ▮
```

Эта подпрограмма имеет следующие характеристики.

Последовательность вызова: JMP NEXTCHAR.

Состояние при входе: rX = количество символов, которые еще будут использованы; rI6 указывает на следующее слово или rI6 = 0. указывая, что нужно читать новую карту.

Состояние при выходе: rA = следующий непустой символ ввода; rX и rI6 настроены для следующего входа в NEXTCHAR.

Наша первая сопрограмма с именем IN находит символы входной последовательности и количество их повторений. В первый раз ее выполнение начинается с IN1.

```
16  * ПЕРВАЯ СОПРОГРАММА
17  2H      INCA 30          Найден нецифровой символ.
18          JMP  OUT         Отослать его в сопрограмму OUT.
19  IN1     JMP  NEXTCHAR    Получить символ.
20          DECA 30
21          JAN  2B          Это буква?
22          CMPA =10=
23          JGE  2B          Это специальный символ?
24          STA  *+1(0:2)    Найдена цифра n.
25          ENT5 *           rI5 ← n.
26          JMP  NEXTCHAR    Взять следующий символ.
27          JMP  OUT         Отослать его в сопрограмму OUT.
28          DEC5 1           Уменьшить n на 1.
29          J5NN *-2         Повторить в случае необходимости.
30          JMP  IN1         Начать новый цикл. ∎
```

(Напоминаю, что в символьном коде MIX цифры 0–9 имеют коды 30–39.) Эта сопрограмма имеет следующие характеристики.

Последовательность вызова: JMP IN.
Состояние при выходе
(при вызове OUT): rA = следующий символ ввода с соответствующим числом повторений; содержимое rI4 остается таким же, как при входе.
Состояние при входе
(после возврата): содержимое rA, rX, rI5, rI6 должно оставаться неизменным с момента последнего выхода.

Вторая сопрограмма с именем OUT разбивает код на группы по три символа и осуществляет перфорацию. Ее выполнение начинается с OUT1.

```
31  * ВТОРАЯ СОПРОГРАММА
32          ALF                      Постоянная, используемая для пробелов.
33  OUTPUT  ORIG *+16                Буферная область для ответов.
34  PUNCH   EQU  17                  Номер устройства перфорирования.
35  OUT1    ENT4 -16                 Начать новую выходную перфокарту.
36          ENT1 OUTPUT
37          MOVE -1,1(16)            Занести в область вывода пробелы.
38  1H      JMP  IN                  Взять следующий преобразованный символ.
39          STA  OUTPUT+16,4(1:1)    Сохранить его в поле (1:1).
40          CMPA PERIOD              Это "."?
41          JE   9F
42          JMP  IN                  Если нет, взять другой символ.
43          STA  OUTPUT+16,4(2:2)    Сохранить его в поле (2:2).
44          CMPA PERIOD              Это "."?
```

```
45          JE    9F
46          JMP   IN                  Если нет, взять следующий символ
47          STA   OUTPUT+16,4(3:3)    Сохранить его в поле (3 3)
48          CMPA  PERIOD              Это "."?
49          JE    9F
50          INC4  1                   Перейти к следующему слову в буфере вывода
51          J4N   1B                  Конец карты?
52  9H      OUT   OUTPUT(PUNCH)       Если да, перфорировать ее
53          JBUS  *(PUNCH)            Ожидать завершения
54          JNE   OUT1                Вернуться за следующими символами, если
55          HLT                         не появилась "."
56  PERIOD  ALF   ␣␣␣␣.                ▮
```

Эта сопрограмма имеет следующие характеристики

Последовательность вызова `JMP OUT`.

Состояние при выходе (при вызове `IN`): Содержимое регистров rA, rX, rI5, rI6 остается неизменным с момента входа, значение в rI1 может измениться, предыдущий символ записывается в выходные данные

Состояние при входе (при возврате). rA = следующий символ ввода с числом повторений; значение в rI4 не меняется с момента последнего выхода.

Для завершения программы нужно обеспечить связь между сопрограммами (см. (1)) и должным образом выполнить инициализацию Инициализация сопрограмм — это довольно тонкое, хотя и несложное, дело

```
57  * ИНИЦИАЛИЗАЦИЯ И СВЯЗЬ
58  START  ENT6  0       Инициализировать rI6 для NEXTCHAR
59         ENTX  0       Инициализировать rX для NEXTCHAR
60         JMP   OUT1    Начать с OUT (см упр 2)
61  OUT    STJ   INX     Связь сопрограмм
62  OUTX   JMP   OUT1
63  IN     STJ   OUTX
64  INX    JMP   IN1
65         END   START   ▮
```

Теперь программа полностью готова Читателю следует тщательно ее изучить, в особенности обращая внимание на то, как можно независимо написать каждую сопрограмму, считая, что другая сопрограмма — это ее подпрограмма

В приведенной выше программе состояния при входе и при выходе для сопрограмм `IN` и `OUT` идеально согласованы Но в более общем случае нам вряд ли так повезет и, чтобы связать сопрограммы, придется также включить команды загрузки и сохранения соответствующих регистров Например, если бы программа `OUT` изменяла содержимое регистра A, то связь сопрограмм нужно было бы запро-

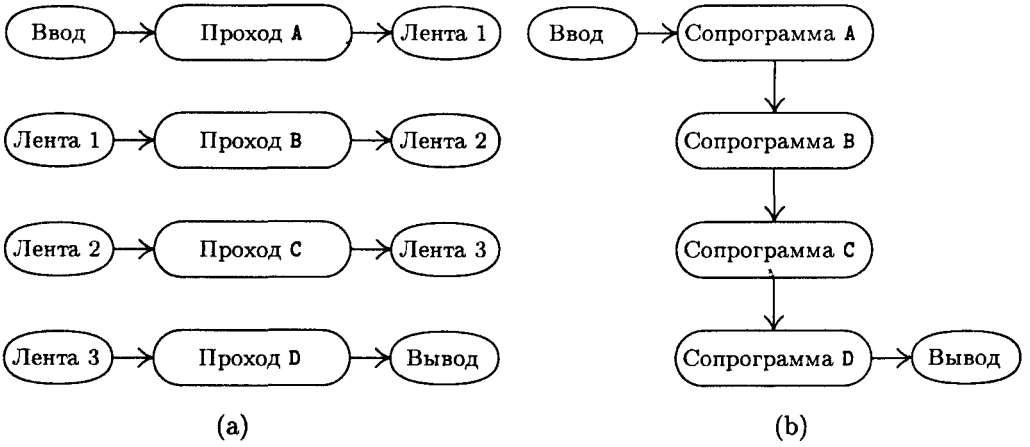


Рис. 22. Проходы: (a) четырехпроходный алгоритм и (b) однопроходный алгоритм.

граммировать следующим образом.

```
OUT   STJ   INX
      STA   HOLDA   Сохранить A при выходе из IN.
OUTX  JMP   OUT1
IN    STJ   OUTX                                        (4)
      LDA   HOLDA   Восстановить A при выходе из OUT.
INX   JMP   IN1      ▮
```

Существует важная связь между сопрограммами и *многопроходными алгоритмами*. Например, описанный выше процесс преобразования можно выполнить за два отдельных прохода. Можно сначала выполнить только сопрограмму IN, применяя ее ко всему вводу и записывая каждый символ с соответствующим числом повторений на магнитную ленту, а затем перемотать ленту в начало и выполнить только сопрограмму OUT, выбирая символы с ленты группами по три. Такой процесс называется двухпроходным. (На интуитивном уровне термин "проход" воспринимается как полный просмотр входных данных. Это определение является неточным, и во многих алгоритмах совсем не очевидно, чему равно число выполненных проходов. Но, несмотря на некоторую неопределенность данного термина, полезно понять его на интуитивном уровне.)

На рис. 22, (a) показан четырехпроходный процесс. Во многих случаях будет оказываться, что такой же процесс можно выполнить только за один проход, как показано в части (b) рисунка, если заменить сопрограммами A, B, C, D соответствующие проходы A, B, C, D. Сопрограмма A вызывает B, после того как во время прохода A элемент выходных данных записан на ленту 1. Сопрограмма B вызывает A, после того как во время прохода B элемент входных данных считан с ленты 1. Сопрограмма B вызывает C, после того как во время прохода B элемент выходных данных записан на ленту 2 и т. д. Пользователи UNIX® узнают в этом "канал", который обозначается как ПроходA | ПроходB | ПроходC | ПроходD. Программы, соответствующие проходам B, C и D, иногда называются фильтрами.

И наоборот, процесс, выполняемый n сопрограммами, часто можно преобразовать в n-проходный процесс. Именно из-за этого соответствия имеет смысл провести сравнительный анализ многопроходных и однопроходных алгоритмов.

a) *Психологическое отличие* Для одной и той же задачи, как правило, проще создать и понять многопроходный алгоритм, чем однопроходный. Процесс, разделенный на ряд небольших шагов, выполняющихся один после другого, понять намного легче, чем запутанную процедуру, в которой множество преобразований выполняется одновременно.

Кроме того, если нужно написать большую серьезную программу, над которой будет совместно работать многочисленная группа разработчиков, то многопроходный алгоритм поможет естественным путем распределить работу между участниками проекта.

Но такие преимущества многопроходного алгоритма присущи и сопрограммам, так как каждую из них можно написать, по существу, отдельно от других, а связь между сопрограммами превратит явно многопроходный алгоритм в однопроходный процесс.

b) *Разница во времени выполнения.* В однопроходном алгоритме не нужно тратить время на то, чтобы упаковать, записать, прочитать и распаковать промежуточные данные, которые передаются между проходами (например, информация на лентах на рис. 22). Поэтому однопроходный алгоритм выполняется быстрее.

c) *Различие в объеме памяти.* Для однопроходного алгоритма необходимо хранить в памяти все программы одновременно, в то время как для многопроходного алгоритма можно сохранять в памяти только по одной программе. Этот фактор может отразиться на скорости выполнения программы даже в большей степени, чем фактор, указанный в п. (b). Например, многие компьютеры имеют ограниченный объем "быстродействующей памяти" и большой объем "медленной памяти". Если каждый проход поместится в быстрой памяти, то вся программа выполнится значительно быстрее, чем в случае использования сопрограмм в одном проходе (так как при использовании сопрограмм большинство программ, скорее всего, будут помещены в "медленную память" либо будут много раз загружаться и выгружаться из быстрой памяти).

Иногда возникает необходимость в разработке алгоритмов сразу для нескольких компьютерных конфигураций, одни из которых имеют больший объем памяти, чем другие. В таких случаях можно написать программу в виде набора сопрограмм и поставить число проходов в зависимость от размера памяти: загрузить сразу столько сопрограмм, сколько поместится, а подачу данных по недостающим связям обеспечить с помощью подпрограмм ввода или вывода.

Несмотря на важность связи между сопрограммами и проходами, нужно иметь в виду, что программу, написанную в виде набора сопрограмм, не всегда можно представить в виде многопроходного алгоритма. Если сопрограмма `B` получает входные данные от `A` и отправляет обратно ключевую информацию сопрограмме `A` (как в приведенном выше примере программы игры в шахматы), то последовательность таких действий нельзя преобразовать в проход `A`, за которым следует проход `B`.

И наоборот, ясно, что некоторые многопроходные алгоритмы нельзя преобразовать в сопрограммы. Отдельные алгоритмы являются многопроходными по своей

сути; например, для второго прохода может требоваться совокупная информация от первого прохода (скажем, общее число появлений некоторого слова во вводе). В этом отношении показательна следующая старая шутка.

В автобусе. Маленькая старушка: "Мальчик, ты не подскажешь мне, когда нужно выходить, чтобы попасть на Пасадена-Стрит?".

Мальчик: "Просто следите за мной и выходите за две остановки перед тем, как выйду я".

(Шутка заключается в том, что мальчик предлагает старушке двухпроходный алгоритм.)

Вот пока и все, что касается многопроходных алгоритмов. С другими примерами сопрограмм мы будем встречаться в различных разделах книги, например они будут частью буферных схем в разделе 1.4.4. Сопрограммы играют также важную роль в моделировании дискретных систем (см. раздел 2.2.5). Важная идея *репликационных сопрограмм* обсуждается в главе 8, а некоторые интересные примеры применения этой идеи можно найти в главе 10.

УПРАЖНЕНИЯ

1. [*10*] Объясните, почему автору трудно найти небольшие простые примеры сопрограмм.

▶ **2.** [*20*] В программе, приведенной в тексте раздела, первой запускается сопрограмма `OUT`. Что произойдет, если первой будет выполняться сопрограмма `IN`, т. е. если в строке 60 поменять "`JMP OUT1`" на "`JMP IN1`"?

3. [*20*] Истинно ли утверждение "Все три команды "`CMPA PERIOD`" из программы `OUT` можно опустить, и программа по-прежнему будет работать"? (Будьте внимательны.)

4. [*20*] Покажите, как связь между сопрограммами, аналогичную (1), можно реализовать на реальных компьютерах, которые вы хорошо знаете.

5. [*15*] Предположим, для обеих сопрограмм `IN` и `OUT` нужно, чтобы в промежутке между входом и выходом содержимое регистра A оставалось неизменным. Другими словами, предположим, что в каждом случае появления команды "`JMP IN`" внутри сопрограммы `OUT` содержимое регистра A должно оставаться неизменным до возврата управления следующей строке. Аналогичное предположение делается по отношению к команде "`JMP OUT`" внутри сопрограммы `IN`. Как в этом случае должна выглядеть связь между сопрограммами? (Ср. с (4).)

▶ **6.** [*22*] Напишите команды связи между сопрограммами, аналогичные (1), для случая *трех* сопрограмм, `A`, `B` и `C`, каждая из которых может вызывать любую из двух других. (В каждом случае активизации сопрограммы выполнение начинается с того места, где оно закончилось в прошлый раз.)

▶ **7.** [*30*] Напишите программу для `MIX`, которая выполняет преобразование, *обратное* тому, которое осуществляет программа из текста раздела, т. е. ваша программа должна получать на входе перфокарты типа (3), а на выходе выдавать перфокарты типа (2). На выходе должна быть настолько короткая строка символов, насколько это возможно, поэтому нуль перед буквой `Z` в (2) на этот раз не должен быть перенесен из (3).

1.4.3. Программы-интерпретаторы

В этом разделе будет рассмотрен один из распространенных типов компьютерных программ — *программы-интерпретаторы* (которые для краткости называют просто *интерпретаторами*). Интерпретатор — это компьютерная программа, которая выполняет команды другой программы, написанной на некотором машинно-ориентированном языке. Под машинно-ориентированным языком подразумевается такой способ представления команд, который предполагает использование в командах кодов операций, адресов и т. д. (Это определение, как и большинство определений современных компьютерных терминов, не является точным, да оно и не должно быть таким; нельзя однозначно определить, какие программы являются интерпретаторами, а какие — нет.)

Исторически первые интерпретаторы являлись оболочками машинно-ориентированных языков, специально предназначенными для упрощения процесса программирования. Такими языками было гораздо легче пользоваться, чем настоящими машинными языками. Возникновение символических языков программирования привело к тому, что отпала необходимость в программах-интерпретаторах такого типа, но это никоим образом не означает, что интерпретаторы начали постепенно исчезать. Наоборот, их продолжали применять и теперь применяют настолько широко, что эффективное использование программ-интерпретаторов можно считать одной из главных характеристик современного программирования. Новые сферы применения интерпретаторов появились, в основном, по следующим причинам:

a) на машинно-ориентированном языке можно представить сложную последовательность решений и действий в компактном и удобном виде;

b) такое представление обеспечивает прекрасный способ передачи информации между проходами в многопроходном процессе.

В подобных случаях разрабатываются машинно-ориентированные языки специального назначения для использования в конкретной программе и программы, написанные на этих языках, часто генерируются только компьютерами. (Современные квалифицированные программисты являются также хорошими разработчиками машин, поскольку они не только создают программу-интерпретатор, но и определяют *виртуальную машину*, язык которой необходимо интерпретировать.)

Принцип интерпретирования имеет еще одно дополнительное преимущество, которое состоит в относительной независимости от машины. Это означает, что при переходе от одного компьютера к другому необходимо переписать только интерпретатор. Более того, полезные средства отладки можно легко встроить в интерпретирующую систему.

Примеры интерпретаторов типа (a) приводятся ниже в нескольких разделах данного многотомника (например, рекурсивный интерпретатор — в главе 8 и машина для синтаксического анализа — в главе 10). Во многих ситуациях, как правило, приходится иметь дело с большим числом аналогичных задач, для которых не удается придумать описывающую их простую общую схему.

Рассмотрим такой пример. Предположим, необходимо написать алгебраический компилятор, с помощью которого можно генерировать эффективные команды машинного языка для сложения двух величин. Эти величины могут принадлежать одному из десяти классов (константы, простые переменные, рабочие ячейки, ин-

дексные переменные, содержимое аккумулятора или индексного регистра, числа с фиксированной или плавающей точкой и т. д.). Если подсчитать количество комбинаций всех пар, то получится 100 различных случаев. И для того чтобы в каждом случае правильно выполнить операцию, понадобится длинная программа. Для решения данной задачи методом интерпретирования нужно изобрести специальный язык, "команды" которого помещались бы в одном байте. Затем необходимо просто подготовить таблицу из 100 "программ" на этом языке, таких, чтобы каждая программа идеально помещалась в одном слове. Идея состоит в том, чтобы выбирать из таблицы соответствующий элемент-программу и выполнять ее. Это простой и эффективный метод.

Пример интерпретатора типа (b) приведен в статье Д. Э. Кнута (D. E. Knuth), "Computer-Drawn Flowcharts", *CACM* **6** (1963), 555–563. В многопроходной программе предыдущие проходы должны передавать информацию последующим. Наиболее эффективным средством передачи этой информации следующему проходу является набор команд на машинно-ориентированном языке. Тогда последующий проход — это ни что иное, как программа-интерпретатор специального назначения, а предыдущий проход — это "компилятор" специального назначения. Такую философию многопроходного процесса можно охарактеризовать следующим образом: мы, по возможности, *рассказываем* последующему проходу, что нужно делать, а не просто передаем ему набор фактов и просим *понять*, что необходимо сделать.

Другой пример интерпретатора типа (b) связан с компиляторами для специальных языков. Если в язык включено много возможностей, которые достаточно просто можно реализовать только в виде подпрограмм, то полученные в результате объектные программы будут представлять собой очень длинные последовательности вызовов подпрограмм. Это может случиться, например, если язык предназначен, главным образом, для выполнения арифметических действий с высокой точностью. В подобном случае объектная программа будет значительно короче, если ее написать на интерпретируемом языке. Например, в книге B. Randell, L. J. Russell, *ALGOL 60 Implementation* (New York: Academic Press, 1964) описывается компилятор, выполняющий трансляцию с языка ALGOL 60 на интерпретируемый язык, а также рассказывается об интерпретаторе для этого языка. В работе Arthur Evans, Jr., "An ALGOL 60 Compiler", *Ann. Rev. Auto. Programming* **4** (1964), 87–124, приводятся также примеры программ-интерпретаторов, которые используются *во внутренней* структуре компилятора. Повсеместное распространение микрокомпьютеров и интегральных микросхем специального назначения сделало этот метод интерпретирования еще более ценным.

Написанная на TEX программа, с помощью которой были созданы страницы этой книги, преобразовала файл, содержащий текст настоящего раздела, в интерпретируемый язык. Этот язык, который называется форматом `DVI`, был разработан Д. Р. Фучсом (D. R. Fuchs) в 1979 году. [См. D. E. Knuth, *TEX: The Program* (Reading, Mass.: Addison-Wesley, 1986), Part 31.] `DVI`-файл, созданный TEX, затем был обработан интерпретатором `dvips`, который написал Т. Г. Рокики (T. G. Rokicki), и преобразован в файл команд на другом интерпретируемом языке под названием PostScript® [Adobe Systems Inc., *PostScript Language Reference Manual*, 2nd edition (Reading, Mass.: Addison-Wesley, 1990)]. Этот PostScript-файл был отослан в издательство, где его распечатали на фотонаборной машине, в которой для получения

печатных пластин используется PostScript-интерпретатор. Такая трехпроходная операция является наглядной иллюстрацией интерпретаторов типа (b); в сам TEX также включен небольшой интерпретатор типа (a), предназначенный для обработки так называемой лигатуры и выполнения кернинга для символов каждого шрифта, встречающегося в тексте [*TEX: The Program*, §545].

На программу, написанную на интерпретируемом языке, можно посмотреть и с другой точки зрения. Ее можно считать набором вызовов подпрограмм, следующих один за другим. Подобную программу можно легко развернуть в длинную последовательность вызовов подпрограмм, и наоборот: такую последовательность обычно можно свернуть в набор команд, который легко интерпретируется. Итак, к преимуществам методов интерпретирования относятся компактность представления, машинная независимость и расширенные возможности диагностики Во многих случаях интерпретатор можно написать так, чтобы затраты времени на интерпретацию самого кода и на переход к нужной программе были незначительны.

1.4.3.1. Имитатор MIX. Если язык, который должен обрабатываться интерпретатором, является машинным языком другого компьютера, то этот интерпретатор обычно называют *имитатором* (а иногда *эмулятором*).

По мнению автора, на написание таких имитаторов потрачено слишком много времени работы программистов, а на их использование — слишком много компьютерного времени. Мотивы создания имитаторов очень просты Например, начальник компьютерного отдела покупает новую машину и хочет по-прежнему использовать на ней программы, написанные для старой машины (вместо того чтобы переписать их с учетом особенностей новой машины). Но обычно это стоит дороже и дает худшие результаты, чем в случае, когда временно нанимается группа программистов и перед ней ставится задача выполнить перепрограммирование. Например, однажды автор принимал участие в подобном проекте и в первоначальной программе, которая использовалась в течение нескольких лет, была обнаружена серьезная ошибка. Мало того что новая программа давала правильные результаты, она еще и работала в пять раз быстрее старой! (Не все имитаторы плохи. Например, для компьютерной фирмы-производителя обычно очень полезно сымитировать новую машину еще до того, как она будет запущена в производство, чтобы программное обеспечение для нее можно было разработать как можно скорее. Но это очень узкая область применения имитаторов.) В качестве яркого примера неэффективного использования имитаторов компьютеров можно привести подлинную историю о машине A, имитирующей машину B, на которой работает программа, имитирующая машину C! Данный способ приводит к тому, что большой и дорогой компьютер дает худшие результаты по сравнению со своим более дешевым собратом.

Ввиду всего вышесказанного возникает вопрос, почему же этот имитатор "поднял свою уродливую голову" в данной книге? На это есть две причины.

a) Имитатор, который будет описан ниже, — это хороший пример типичной программы-интерпретатора. Здесь проиллюстрированы основные методы, использующиеся в интерпретаторах, и, кроме того, демонстрируется применение подпрограмм в достаточно длинной программе.

b) Будет рассмотрен имитатор компьютера MIX, написанный на языке MIX (подумать только!). Это облегчит написание имитаторов MIX для большинства компью-

теров, подобных MIX; в коде нашей программы мы намеренно избегали широкого использования возможностей, присущих исключительно MIX. Имитатор MIX пригодится вам в качестве наглядного пособия к этой книге и, возможно, к другим.

Компьютерные имитаторы, описываемые в настоящем разделе, следует отличать от *имитаторов дискретных систем.* Имитаторы дискретных систем — это важные программы, которые будут обсуждаться в разделе 2.2.5.

А теперь вернемся к задаче написания имитатора MIX. Входными данными для нашей программы будут последовательность команд MIX и данные, сохраненные в ячейках 0000–3499. Мы в точности сымитируем работу аппаратного обеспечения MIX и сделаем вид, что MIX сам интерпретирует эти команды. Таким образом мы попытаемся реализовать спецификации, которые были определены в разделе 1.3.1. В нашей программе, например, используется переменная AREG, с помощью которой сохраняется модуль значения, содержащегося в имитируемом регистре A; другая переменная, SIGNA, используется для хранения соответствующего знака. С помощью переменной CLOCK ведется учет количества единиц имитируемого времени MIX, затраченного на выполнение имитируемой программы.

Нумерация таких команд MIX, как LDA, LD1, ..., LDX и других подобных команд, подсказывает нам, что сохранять имитируемое содержимое этих регистров в последовательных ячейках нужно так:

AREG, I1REG, I2REG, I3REG, I4REG, I5REG, I6REG, XREG, JREG, ZERO.

Здесь ZERO — это "регистр", постоянно заполненный нулями. Позиции JREG и ZERO выбраны в соответствии с кодами операций команд STJ и STZ.

Согласно нашей философии написания имитатора, т. е. ориентируясь на любой компьютер, а не только на MIX, будем рассматривать знаки как независимые части регистра. Например, во многих компьютерах нельзя представить число "минус нуль", а в MIX можно, поэтому в данной программе знаки всегда будут обрабатываться особым образом. В ячейках AREG, I1REG, ..., ZERO всегда будут храниться абсолютные значения величин, содержащихся в соответствующих регистрах. В другом наборе ячеек, использующемся в данной программе, (SIGNA, SIGN1, ..., SIGNZ), будут содержаться значения $+1$ или -1, в зависимости от знака соответствующего регистра (т. е. "плюс" это или "минус").

В программе-интерпретаторе, как правило, есть раздел, который представляет собой орган центрального управления. Он вступает в действие между интерпретируемыми командами. В нашем случае после выполнения каждой сымитированной команды программа переходит к ячейке CYCLE.

Управляющая программа выполняет одинаковые для всех команд действия; она разделяет команду на составные части и помещает эти части в такие места, откуда их будет удобно выбирать для дальнейшего использования. В приведенной ниже программе используются следующие установки:

rI6 — адрес ячейки, в которой сохраняется следующая команда;
rI5 — M (адрес текущей команды с учетом индексирования);
rI4 — код операции текущей команды;
rI3 — F-поле текущей команды;
INST — текущая команда.

Программа М.

```
001  * ИМИТАТОР MIX
002          ORIG 3500          Имитируемая память, начиная с 0000 и далее.
003  BEGIN   STZ  TIME(0:2)
004          STZ  OVTOG         OVTOG — имитируемый флаг переполнения.
005          STZ  COMPI         COMPI, ±1 или 0, — флаг сравнения
006          ENT6 0             Взять первую команду из нулевой ячейки
007  CYCLE   LDA  CLOCK         Начало выполнения управляющей программы.
008  TIME    INCA 0             Указывает на ячейку, содержащую время вы-
009          STA  CLOCK           полнения предыдущей команды (строка 033)
010          LDA  0,6           rA ← имитируемая команда
011          STA  INST
012          INC6 1             Увеличить значение счетчика адреса
013          LDX  INST(1:2)     Взять абсолютное значение адреса
014          SLAX 5             Присоединить знак к адресу
015          STA  M
016          LD2  INST(3:3)     Проверить поле индекса
017          J2Z  1F            Это нуль?
018          DEC2 6
019          J2P  INDEXERROR    Определен недопустимый индекс?
020          LDA  SIGN6,2       Взять знак индексного регистра
021          LDX  I6REG,2       Взять абсолютное значение индексного регистра.
022          SLAX 5             Присоединить знак
023          ADD  M             Сложение с учетом знаков для индексирования.
024          CMPA ZERO(1:3)     Результат слишком велик?
025          JNE  ADDRERROR     Если да, имитировать ошибку
026          STA  M             В противном случае адрес найден
027  1H      LD3  INST(4:4)     rI3 ← F-поле
028          LD5  M             rI5 ← M
029          LD4  INST(5:5)     rI4 ← C-поле
030          DEC4 63
031          J4P  OPERROR       Код операции ≥ 64?
032          LDA  OPTABLE,4(4:4) Взять из таблицы время выполнения
033          STA  TIME(0:2)
034          LD2  OPTABLE,4(0:2) Взять адрес соответствующей программы
035          JNOV 0,2           Перейти к оператору
036          JMP  0,2           (Защита от переполнений ) ▮
```

Советую читателю обратить особое внимание на строки 034–036 "Таблица-переключатель" из 64 операторов — это составная часть имитатора, позволяющая ему быстро переходить к нужной программе для выполнения текущей команды. Это важный метод экономии времени (см упр. 1 3 2–9).

В состоящей из 64 слов таблице-переключателе OPTABLE также хранится время выполнения для различных операторов, содержимое этой таблицы определяется в следующих строках.

```
037          NOP  CYCLE(1)      Таблица кодов операций,
038          ADD  ADD(2)            ее типичные элементы имеют вид
039          SUB  SUB(2)            "ОП программа(время)"
040          MUL  MUL(10)
```

```
041          DIV  DIV(12)
042          HLT  SPEC(1)
043          SLA  SHIFT(2)
044          MOVE MOVE(1)
045          LDA  LOAD(2)
046          LD1  LOAD,1(2)
             ...
051          LD6  LOAD,1(2)
052          LDX  LOAD(2)
053          LDAN LOADN(2)
054          LD1N LOADN,1(2)
             ...
060          LDXN LOADN(2)
061          STA  STORE(2)
             ..
069          STJ  STORE(2)
070          STZ  STORE(2)
071          JBUS JBUS(1)
072          IOC  IOC(1)
073          IN   IN(1)
074          OUT  OUT(1)
075          JRED JRED(1)
076          JMP  JUMP(1)
077          JAP  REGJUMP(1)
             ...
084          JXP  REGJUMP(1)
085          INCA ADDROP(1)
086          INC1 ADDROP,1(1)
             ...
092          INCX ADDROP(1)
093          CMPA COMPARE(2)

100  OPTABLE CMPX COMPARE(2)
```

(Элементы, соответствующие операторам LDi, LDiN и INCi, имеют дополнительную запись ",1", чтобы сделать поле (3:3) ненулевым. Это используется ниже, в строках 289 и 290, для указания того, что после имитации данных операций необходимо проверить размер величины, содержащейся в соответствующем индексном регистре.)

В следующей части программы-имитатора просто перечисляются ячейки, которые используются для хранения содержимого имитируемых регистров.

```
101  AREG    CON  0   Абсолютное значение регистра A.
102  I1REG   CON  0   Абсолютное значение индексных регистров.
             ...
107  I6REG   CON  0
108  XREG    CON  0   Абсолютное значение регистра X.
109  JREG    CON  0   Абсолютное значение регистра J.
110  ZERO    CON  0   Нулевая константа для "STZ".
111  SIGNA   CON  1   Знак регистра A.
112  SIGN1   CON  1   Знаки индексных регистров.
             ...
```

```
117  SIGN6   CON  1
118  SIGNX   CON  1   Знак регистра X.
119  SIGNJ   CON  1   Знак регистра J.
120  SIGNZ   CON  1   Знак, сохраняемый "STZ".
121  INST    CON  0   Имитируемая команда.
122  COMPI   CON  0   Флаг сравнения
123  OVTOG   CON  0   Флаг переполнения.
124  CLOCK   CON  0   Имитируемое время выполнения.  ▌
```

А теперь рассмотрим три подпрограммы, используемые имитатором. Первой идет подпрограмма MEMORY.

Последовательность вызова: `JMP MEMORY`.
Состояние при входе: rI5 = допустимый адрес памяти (в противном случае подпрограмма перейдет к MEMERROR).
Состояние при выходе: rX = знак слова в ячейке памяти rI5; rA = абсолютное значение слова в ячейке памяти rI5.

```
125  * ПОДПРОГРАММЫ
126  MEMORY  STJ   9F          Подпрограмма выборки из памяти.
127          J5N   MEMERROR
128          CMP5  =BEGIN=     Имитируемая память находится
129          JGE   MEMERROR        в ячейках с 0000 до BEGIN - 1.
130          LDX   0,5
131          ENTA  1
132          SRAX  5           rX ← знак слова.
133          LDA   0,5(1:5)    rA ← абсолютное значение слова.
134  9H      JMP   *           Выход.  ▌
```

Подпрограмма FCHECK обрабатывает спецификацию частичного поля и проверяет, имеет ли оно вид $8L + R$, где $L \le R \le 5$.

Последовательность вызова: `JMP FCHECK`.
Состояние при входе: rI3 = допустимая спецификация поля (в противном случае подпрограмма перейдет к FERROR).
Состояние при выходе: rA = rI1 = L, rX = R.

```
135  FCHECK  STJ   9F          Подпрограмма проверки поля.
136          ENTA  0
137          ENTX  0,3         rAX ← спецификация поля.
138          DIV   =8=         rA ← L, rX ← R.
139          CMPX  =5=         R > 5?
140          JG    FERROR
141          STX   R
142          STA   L
143          LD1   L           rI1 ← L.
144          CMPA  R
145  9H      JLE   *           Выйти, если L ≤ R.
146          JMP   FERROR  ▌
```

Последняя подпрограмма, GETV, находит величину V (т. е. соответствующее поле ячейки M), используемую различными операторами MIX, как определено в разделе 1.3.1.

Последовательность вызова: JMP GETV.
Состояние при входе: rI5 = допустимый адрес памяти; rI3 = допустимое поле. (Если оно недопустимо, то регистрируется ошибка, как и выше.)
Состояние при выходе: rA = абсолютное значение V; rX = знак V; rI1 = L; rI2 = −R.
Второй вход: JMP GETAV, используется только в операторах сравнения для выборки поля из регистра.

```
147 GETAV   STJ  9F      Специальный вход (см строку 300).
148         JMP  1F
149 GETV    STJ  9F      Подпрограмма для нахождения V.
150         JMP  FCHECK  Обработать поле и установить rI1 ← L.
151         JMP  MEMORY  rA ← абсолютное значение памяти, rX ← знак.
152 1H      J1Z  2F      Знак включен в поле?
153         ENTX 1       Если нет, установите положительный знак.
154         SLA  -1,1    Обнулите все байты слева
155         SRA  -1,1        от этого поля
156 2H      LD2N R       Сдвиг вправо
157         SRA  5,2         в соответствующую позицию.
158 9H      JMP  *       Выход. ▌
```

А теперь рассмотрим программы для каждого оператора в отдельности. Данные программы приведены здесь для полноты изложения, поэтому читателю стоит изучить лишь некоторые из них, если, конечно, у него нет важных причин для более тщательного изучения. Рекомендуется изучить программы для операторов SUB и JUMP: это наиболее типичные примеры. Обратите внимание на то, как можно аккуратно объединить программы для аналогичных операций, а также на то, как программа JUMP использует другую таблицу-переключатель для управления этим типом перехода.

```
159 * ОТДЕЛЬНЫЕ ОПЕРАТОРЫ
160 ADD     JMP  GETV        Загрузить значение V в rA и rX
161         ENT1 0           Пусть rI1 указывает на регистр A
162         JMP  INC         Перейти к программе "увеличения"
163 SUB     JMP  GETV        Загрузить значение V в rA и rX
164         ENT1 0           Пусть rI1 указывает на регистр A
165         JMP  DEC         Перейти к программе "уменьшения".
166 *
167 MUL     JMP  GETV        Загрузить значение V в rA и rX.
168         CMPX SIGNA       Знаки одинаковы?
169         ENTX 1
170         JE   *+2         Занести в rX знак результата
171         ENNX 1
172         STX  SIGNA       Поместить его в оба имитируемых регистра.
173         STX  SIGNX
```

```
174           MUL   AREG          Перемножить операнды.
175           JMP   STOREAX       Сохранить абсолютные значения.
176  *
177  DIV      LDA   SIGNA         Установить знак остатка.
178           STA   SIGNX
179           JMP   GETV          Загрузить значение V в rA и rX.
180           CMPX  SIGNA         Знаки одинаковы?
181           ENTX  1
182           JE    *+2           Занести в rX знак результата.
183           ENNX  1
184           STX   SIGNA         Поместить его в имитируемый rA.
185           STA   TEMP
186           LDA   AREG          Разделить операнды.
187           LDX   XREG
188           DIV   TEMP
189  STOREAX  STA   AREG          Сохранить абсолютные значения.
190           STX   XREG
191  OVCHECK  JNOV  CYCLE         Только что произошло переполнение?
192           ENTX  1             Если да, установить имитируемый
193           STX   OVTOG               флаг переполнения в положение 1.
194           JMP   CYCLE         Вернуться в управляющую программу.
195  *
196  LOADN    JMP   GETV          Загрузить значение V в rA и rX.
197           ENT1  47,4          rI1 ← C − 16; указывает регистр.
198  LOADN1   STX   TEMP          Сделать знак отрицательным.
199           LDXN  TEMP
200           JMP   LOAD1         Заменить LOADN на LOAD.
201  LOAD     JMP   GETV          Загрузить значение V в rA и rX.
202           ENT1  55,4          rI1 ← C − 8, указывает регистр.
203  LOAD1    STA   AREG,1        Сохранить абсолютные значения.
204           STX   SIGNA,1       Сохранить знак.
205           JMP   SIZECHK       Проверить, не слишком ли велико
206  *                                 абсолютное значение.
207  STORE    JMP   FCHECK        rI1 ← L.
208           JMP   MEMORY        Взять содержимое ячейки памяти.
209           J1P   1F            Знак включен в поле?
210           ENT1  1             Если да, заменить L на 1
211           LDX   SIGNA+39,4          и "сохранить" знак регистра.
212  1H       LD2N  R             rI2 ← −R.
213           SRAX  5,2           Сохранить область справа от поля.
214           LDA   AREG+39,4     Вставить регистр в поле.
215           SLAX  5,2
216           ENN2  0,1           rI2 ← −L.
217           SRAX  6,2
218           LDA   0,5           Восстановить область слева от поля.
219           SRA   6,2
220           SRAX  -1,1          Присоединить знак.
221           STX   0,5           Сохранить в памяти.
222           JMP   CYCLE         Вернуться в управляющую программу.
223  *
```

```
224  JUMP     DEC3 9              Операторы перехода.
225           J3P  FERROR         F слишком велико?
226           LDA  COMPI          rA ← флаг сравнения.
227           JMP  JTABLE,3       Перейти к соответствующей программе.
228  JMP      ST6  JREG           Установить имитируемый регистр J.
229           JMP  JSJ
230           JMP  JOV
231           JMP  JNOV
232           JMP  LS
233           JMP  EQ
234           JMP  GR
235           JMP  GE
236           JMP  NE
237  JTABLE   JMP  LE             Конец таблицы перехода.
238  JOV      LDX  OVTOG          Проверить, делать ли переход по
239           JMP  *+3                переполнению.
240  JNOV     LDX  OVTOG
241           DECX 1              Взять дополнение флага переполнения.
242           STZ  OVTOG          Отключить флаг переполнения.
243           JXNZ JMP            Перейти.
244           JMP  CYCLE          Не переходить.
245  LE       JAZ  JMP            Перейти, если в rA — нуль или отрицат. число.
246  LS       JAN  JMP            Перейти, если в rA — отрицат. число.
247           JMP  CYCLE          Не переходить.
248  NE       JAN  JMP            Перейти, если в rA — отрицат. или положит.
                                      число.
249  GR       JAP  JMP            Перейти, если в rA — положит. число.
250           JMP  CYCLE          Не переходить.
251  GE       JAP  JMP            Перейти, если в rA — положит. число или нуль.
252  EQ       JAZ  JMP            Перейти, если в rA — нуль.
253           JMP  CYCLE          Не переходить.
254  JSJ      JMP  MEMORY         Проверить допустимость адресов памяти.
255           ENT6 0,5            Имитировать переход.
256           JMP  CYCLE          Вернуться в главную управляющую программу.
257  *
258  REGJUMP  LDA  AREG+23,4      Регистровые переходы.
259           JAZ  *+2            В регистре нуль?
260           LDA  SIGNA+23,4     Если нет, поместить знак в rA.
261           DEC3 5
262           J3NP JTABLE,3       Заменить командой условного перехода, если
263           JMP  FERROR             F-спецификация не слишком велика.
264  *
265  ADDROP   DEC3 3              Операторы пересылки адреса.
266           J3P  FERROR         F слишком велико?
267           ENTX 0,5
268           JXNZ *+2            Найти знак M.
269           LDX  INST
270           ENTA 1
271           SRAX 5              rX ← знак M.
272           LDA  M(1:5)         rA ← абсолютное значение M.
```

```
273             ENT1  15,4              rI1 указывает регистр.
274             JMP   1F,3               Четырехсторонний переход.
275             JMP   INC                Увеличить.
276             JMP   DEC                Уменьшить.
277             JMP   LOAD1              Занести.
278   1H        JMP   LOADN1             Занести отрицательное число.
279   DEC       STX   TEMP               Изменить знак на противоположный.
280             LDXN  TEMP               Заменить DEC на INC.
281   INC       CMPX  SIGNA,1            Программа сложения.
282             JE    1F                 Знаки одинаковы?
283             SUB   AREG,1             Нет; вычесть абсолютные значения.
284             JANP  2F                 Нужно ли изменить знак?
285             STX   SIGNA,1            Изменить знак регистра.
286             JMP   2F
287   1H        ADD   AREG,1             Сложить абсолютные значения.
288   2H        STA   AREG,1(1:5)        Сохранить абсолютное значение результата.
289   SIZECHK   LD1   OPTABLE,4(3:3)     Только что был загружен
290             J1Z   OVCHECK                индексный регистр?
291             CMPA  ZERO(1:3)          Если да, убедиться, что результат
292             JE    CYCLE                  помещается в двух байтах.
293             JMP   SIZEERROR
294   *
295   COMPARE   JMP   GETV               Загрузить значение V в rA и rX.
296             SRAX  5                  Присоединить знак.
297             STX   V
298             LDA   XREG,4             Взять поле F соответствующего регистра.
299             LDX   SIGNX,4
300             JMP   GETAV
301             SRAX  5                  Присоединить знак.
302             CMPX  V                  Сравнить (заметьте, что −0 = +0).
303             STZ   COMPI              Установить флаг сравнения
304             JE    CYCLE                  в положение 0, +1
305             ENTA  1                      либо −1.
306             JG    *+2
307             ENNA  1
308             STA   COMPI
309             JMP   CYCLE              Вернуться в управляющую программу.
310   *
311             END   BEGIN
```

В приведенном выше коде используется одно хитрое правило, сформулированное в разделе 1.3.1: команда "ENTA -0" загружает минус нуль в регистр A, так же как команда "ENTA -5,1", когда в индексном регистре 1 содержится значение +5. В общем случае, если M равно нулю, команда ENTA загружает знак этой команды, а ENNA загружает противоположный знак. Когда автор писал первый черновик раздела 1.3.1, он не придал особого значения определению этого условия. Такие вопросы обычно возникают только при написании компьютерной программы, в которой используются данные правила.

Несмотря на свои размеры, приведенная выше программа является неполной в нескольких отношениях.

a) Она не распознает операций с плавающей точкой.
b) Программирование кодов операций 5–7 оставлено для упражнения.
c) Программирование операторов ввода-вывода оставлено для упражнения.
d) Не предусмотрены средства загрузки имитируемых программ (см. упр. 4).
e) Не включены программы

INDEXERROR, ADDRERROR, OPERROR, MEMERROR, FERROR, SIZEERROR.

Они предназначены для обработки ошибок, обнаруженных в имитируемой программе.
f) Не предусмотрены средства диагностики. (Хороший имитатор должен, например, быть способным распечатать содержимое регистров во время выполнения программы.)

УПРАЖНЕНИЯ

1. [*14*] Изучите все варианты применения подпрограммы FCHECK в программе-имитаторе. Можете ли вы предложить более удачный способ организации программы? (См. шаг 3 в конце раздела 1.4.1.)

2. [*20*] Напишите программу SHIFT, которой не хватает в программе, приведенной в тексте (код операции 6).

3. [*22*] Напишите программу MOVE, которой не хватает в программе, приведенной в тексте (код операции 7).

4. [*14*] Измените программу в тексте раздела, чтобы она начиналась так, как будто была нажата "кнопка GO" машины MIX (см. упр. 1.3.1–26).

▸ **5.** [*24*] Определите, сколько времени потребуется для имитирования операторов LDA и ENTA *по сравнению с реальным временем, которое* MIX *затратит на их непосредственное* выполнение.

6. [*28*] Напишите программы для операторов ввода-вывода JBUS, IOC, IN, OUT и JRED, которых не хватает в программе, приведенной в тексте раздела, разрешив использовать только устройства 16 и 18. Предполагается, что операции "читать перфокарту" и "перейти к новой странице" занимают $T = 10000u$, в то время как операция "печатать строку" занимает $T = 7500u$. [*Замечание.* Опыт показывает, что команду JBUS следует имитировать, рассматривая "JBUS *" как частный случай; иначе имитатор, похоже, остановится!]

▸ **7.** [*32*] Модифицируйте решение предыдущего упражнения таким образом, чтобы выполнение команды IN или OUT не приводило к немедленной передаче входных-выходных данных. Такая передача должна происходить только после того, как пройдет примерно половина времени, которое необходимо для имитируемых устройств. (Это позволит избежать распространенной студенческой ошибки, когда команды IN и OUT используются неправильно.)

8. [*20*] Истинно или ложно следующее утверждение: "Каждый раз при выполнении строки 010 программы-имитатора выполняется неравенство $0 \le$ rI6 $<$ BEGIN"?

***1.4.3.2. Программы трассировки.** Когда машина имитируется на самой себе (как в предыдущем разделе MIX имитировался на MIX), получается частный случай имитатора, который называется программой *трассировки* или *слежения*. Подобные программы иногда используются при отладке, так как позволяют распечатать пошаговый отчет о поведении имитируемой программы.

В предыдущем разделе программа была написана так, как будто MIX имитировался на другом компьютере. Для программ трассировки используется совершенно другой подход; обычно мы позволяем, чтобы регистры представляли сами себя, а операторы выполняли сами себя, т. е. позволяем машине самой выполнять большинство команд. Основное исключение представляет команда перехода или условного перехода, которую нельзя выполнять, не модифицируя, так как программа трассировки должна сохранять общий контроль. Каждой машине присущи уникальные, свойственные только ей особенности, и это значительно усложняет трассировку. Для машины MIX самая трудная и интересная проблема связана с регистром J.

Приведенная ниже программа трассировки запускается, когда главная программа переходит к ячейке ENTER. При этом в регистре J содержится адрес, с которого трассировка должна *начаться*, а в регистре X — адрес, где она должна *закончиться*. Это интересная программа, заслуживающая внимательного изучения.

```
01  * TRACE ROUTINE
02  ENTER   STX   TEST(0:2)      Установить адрес выхода.
03          STX   LEAVEX(0:2)
04          STA   AREG           Сохранить содержимое rA.
05          STJ   JREG           Сохранить содержимое rJ.
06          LDA   JREG(0:2)      Взять начальный адрес для трассировки.
07  CYCLE   STA   PREG(0:2)      Сохранить адрес следующей команды.
08  TEST    DECA  *              Это адрес выхода?
09          JAZ   LEAVE
10  PREG    LDA   *              Взять следующую команду.
11          STA   INST           Скопировать ее.
12          SRA   2
13          STA   INST1(0:3)     Сохранить этот адрес и индексы.
14          LDA   INST(5:5)      Взять код операции, C.
15          DECA  38
16          JANN  1F             C ≥ 38 (JRED)?
17          INCA  6
18          JANZ  2F             C ≠ 32 (STJ)?
19          LDA   INST(0:4)
20          STA   *+2(0:4)       Заменить STJ на STA.
21  JREG    ENTA  *              rA ← имитируемое содержимое rJ.
22          STA   *
23          JMP   INCP
24  2H      DECA  2
25          JANZ  2F             C ≠ 34 (JBUS)?
26          JMP   3F
27  1H      DECA  9              Тест для команд перехода.
28          JAP   2F             C > 47 (JXL)?
29  3H      LDA   8F(0:3)        Мы обнаружили команду перехода;
30          STA   INST(0:3)          изменить ее адрес на "JUMP".
31  2H      LDA   AREG           Восстановить регистр A.
32  *                            Во всех регистрах, кроме J, теперь правильные
33  *                                значения по отношению к внешней программе.
34  INST    NOP   *              Команда выполняется.
35          STA   AREG           Снова сохранить регистр A.
36  INCP    LDA   PREG(0:2)      Перейти к следующей команде.
```

```
37          INCA 1
38          JMP  CYCLE
39  8H      JSJ  JUMP          Константа для строк 29 и 40.
40  JUMP    LDA  8B(4:5)       Встретился переход.
41          SUB  INST(4:5)     Это была JSJ?
42          JAZ  *+4
43          LDA  PREG(0:2)     Если нет, обновите имитируемый
44          INCA 1                  регистр J.
45          STA  JREG(0:2)
46  INST1   ENTA *
47          JMP  CYCLE         Перейти к адресу перехода.
48  LEAVE   LDA  AREG          Восстановить регистр A.
49  LEAVEX  JMP  *             Остановить трассировку.
50  AREG    CON  0             Содержимое имитируемого rA.  ▮
```

По поводу программ трассировки в целом и этой в частности нужно отметить следующее.

1) Здесь представлена только самая интересная часть программы трассировки — часть, которая является управляющей во время выполнения другой программы. Чтобы трассировка принесла пользу, необходима также программа записи содержимого регистров, но мы ее не включили. Такая программа, хотя и безусловно важная, отвлекает внимание от более тонких моментов программы трассировки. Поэтому модификации, которые в этой связи необходимо ввести в программу, оставлены для упражнения (см. упр. 2).

2) Занимаемое пространство обычно имеет большее значение, чем время выполнения, т. е. программа должна быть настолько короткой, насколько это возможно. Тогда программа трассировки сможет "сосуществовать" даже с очень большими программами. А время выполнения все равно расходуется на вывод данных.

3) Мы позаботились о том, чтобы избежать изменения содержимого большинства регистров. Фактически программа использует только регистр A машины MIX. Программа трассировки не оказывает влияния ни на флаг сравнения, ни на флаг переполнения. (Чем меньше регистров используется, тем меньшее их число нужно восстанавливать.)

4) Когда происходит переход к ячейке JUMP, необязательно выполнять команду "STA AREG", так как содержимое rA не изменилось.

5) После выхода из программы трассировки регистр J не восстанавливается. В упр. 1 показано, как исправить эту ситуацию.

6) На трассируемую программу налагаются только три ограничения.

a) Она не должна сохранять что-либо в ячейках, используемых программой трассировки.

b) Она не должна использовать выходное устройство, на которое выводится информация о трассировке (например, команда JBUS дала бы неправильное указание).

c) Во время трассировки она будет работать медленнее.

УПРАЖНЕНИЯ

1. [*22*] Модифицируйте программу трассировки так, чтобы при выходе она восстанавливала регистр J. (Можете предполагать, что содержимое регистра J *ненулевое*.)

2. [*26*] Модифицируйте приведенную в тексте программу трассировки таким образом, чтобы перед выполнением каждого шага она записывала на магнитную ленту (устройство 0) следующую информацию.

Слово 1, поле (0:2): адрес ячейки.

Слово 1, поле (4:5): регистр J (перед выполнением).

Слово 1, поле (3:3): 2, если результат сравнения — больше, 1 — если равно, 0 — если меньше; плюс 8, если перед выполнением нет переполнения.

Слово 2: команда.

Слово 3: регистр A (перед выполнением).

Слова 4–9: регистры I1–I6 (перед выполнением).

Слово 10: регистр X (перед выполнением).

Слова 11–100 каждого блока ленты размером 100 слов должны содержать еще девять групп по 10 слов, записанных в том же формате.

3. [*10*] В предыдущем упражнении программа трассировки записывает свои выходные данные на магнитную ленту Объясните, почему это лучше, чем непосредственно распечатать результаты.

▶ **4.** [*25*] Что произойдет, если программа трассировки будет трассировать *саму себя*? Если более конкретно, то выясните, как будет работать программа трассировки, если две команды — `ENTX LEAVEX` и `JMP *+1` — поместить непосредственно перед `ENTER`

5. [*28*] Рассмотрим задачу, аналогичную предыдущей. Пусть две копии программы трассировки помещены в различных местах памяти и настроены так, чтобы трассировать одна другую. Что произойдет?

▶ **6.** [*40*] Напишите программу трассировки, способную трассировать себя в смысле упр 4: она должна в замедленном режиме распечатывать собственные шаги. а *та* программа будет трассировать саму себя в еще *более замедленном* режиме, и так до бесконечности, пока будет хватать памяти.

▶ **7.** [*25*] Подумайте, как написать эффективную программу *трассировки переходов*, которая выдает намного меньше выходных данных, чем обычная программа трассировки. Вместо того чтобы отображать содержимое регистров, программа трассировки переходов просто записывает происходящие переходы Она выдает последовательность пар (x_1, y_1), (x_2, y_2), ... Это означает, что программа перешла из ячейки x_1 в y_1, затем (после выполнения команд из ячеек y_1, $y_1 + 1$, ..., x_2) — из ячейки x_2 в y_2 и т. д. [На основании этой информации следующая программа может восстановить ход выполнения программы и установить, как часто выполнялась каждая команда.]

1.4.4. Ввод и вывод

Вероятно, самое очевидное, чем один компьютер отличается от другого, — это устройства ввода-вывода, а также компьютерные команды, управляющие этими периферийными устройствами. В рамках одной книги невозможно обсудить все проблемы и методы, связанные с данной областью, поэтому мы ограничимся изучением наиболее типичных методов ввода-вывода, применимых для большинства компьютеров. Операторы ввода-вывода машины `MIX` представляют собой нечто среднее между самыми разнообразными средствами, имеющимися в реальных компьютерах.. Чтобы вы могли представить себе операции ввода-вывода на конкретном примере, давайте в этом разделе обсудим проблемы оптимального выполнения операций ввода-вывода в машине `MIX`.

И снова я прошу читателя снисходительно отнестись к тому, что здесь рассматривается анахроничная машина MIX *с ее перфокартами и т. п. Хотя эти устаревшие устройства сегодня полностью вышли из употребления, они по-прежнему могут дать несколько важных уроков. Но, конечно, когда для этой цели будет использоваться компьютер* MMIX*, он преподаст нам их еще лучше.*

Многие пользователи компьютеров полагают, что ввод и вывод на самом деле не являются частью процесса "настоящего" программирования. Считается, что ввод и вывод — это нудные задачи, которые приходится выполнять только для того, чтобы ввести информацию в компьютер и вывести из него полученные результаты. Поэтому средства ввода-вывода компьютера обычно начинают изучать только после знакомства с остальными его возможностями. И часто случается так, что лишь небольшая группа программистов некоторого компьютера вообще что-либо знает о деталях операций ввода-вывода. Такое положение вещей в чем-то даже естественно, поскольку программирование ввода-вывода никогда не было особенно привлекательным. Но до тех пор, пока многие не начнут серьезно относиться к данному предмету, нельзя ожидать, что ситуация изменится. В этом и других разделах (например, в разделе 5.4.6) вы увидите, что в связи с вводом-выводом возникает множество интересных вопросов, а также узнаете о существовании некоторых изящных алгоритмов.

Теперь, пожалуй, нужно сделать небольшое отступление по поводу терминологии. В словарях английского языка слова "input" ("ввод") и "output" ("вывод") раньше приводились только как существительные ("What kind of input are we getting?" — "Какого вида данные мы вводим?"). Но теперь уже вошло в привычку использовать их как прилагательные ("Don't drop the input tape" — "Не сотрите эту ленту с входными данными") и переходные глаголы ("Why did the program output this garbage?" — "Почему программа выводит эту чепуху?"). Вместо комбинированного термина "ввод-вывод" чаще всего используют аббревиатуру "В/В" (на английском — "I/O"). Операцию ввода часто называют *чтением*, а вывода соответственно *записью*. Элементы, составляющие ввод или вывод, обычно называют "данными" (на английском — "data"). В английском языке это слово, строго говоря, является формой множественного числа (как и в русском языке. — *Прим. перев.*), но используется в собирательном смысле, как будто это единственное число ("The data has not been read."). Точно так и слово "information" ("информация") является формой как единственного, так и множественного числа. На этом урок английского закончен.

Предположим, необходимо прочитать данные с магнитной ленты. Оператор IN машины MIX, как описано в разделе 1.3.1, просто *инициирует* процесс ввода, и в то время, пока данные вводятся, компьютер продолжает выполнять следующие команды. Поэтому по команде "IN 1000(5)" начинается считывание 100 слов с накопителя на магнитных лентах под номером 5 в ячейки памяти 1000–1099, но последующие команды программы пока не должны обращаться к этим ячейкам. Программа может считать, что ввод завершен только после того, как (a) инициируется другая операция В/В (IN, OUT или IOC), обращающаяся к устройству 5, или (b) команда условного перехода JBUS(5) или JRED(5) показывает, что устройство 5 больше не "занято".

Поэтому самый простой способ считать блок данных с ленты в ячейки 1000–1099 так, чтобы информация была на месте, — воспользоваться двумя командами:

```
IN 1000(5); JBUS *(5).        (1)
```

Этот элементарный метод применялся в программе из раздела 1.4.2 (см. строки 07–08 и 52–53). Но он слишком неэкономно расходует время работы компьютера, так как большая часть времени, которую можно было бы использовать для вычислений, скажем $1000u$ или даже $10000u$, тратится на многократное повторение команды "JBUS". Если это время использовать для вычислений, то скорость выполнения программы можно удвоить. (См. упр. 4 и 5.)

Один из способов избежать такого "цикла ожидания" — использовать две области памяти для ввода. Можно считывать данные в одну область, в то же время выполняя вычисления над данными в другой. Например, программу можно начать командой

```
IN    2000(5)    Начать чтение первого блока.                    (2)
```

И теперь каждый раз, когда понадобится прочитать блок с ленты, можно дать пять следующих команд.

```
ENT1  1000       Подготовиться к выполнению оператора MOVE.
JBUS  *(5)       Ожидать готовности устройства 5.
MOVE  2000(50)   (2000–2049) → (1000–1049).                      (3)
MOVE  2050(50)   (2050–2099) → (1050–1099).
IN    2000(5)    Начать чтение следующего блока.
```

В целом, это дает такой же эффект, как и применение команды (1), но лента с входными данными остается занятой, пока программа работает над данными, находящимися в ячейках 1000–1099.

Последняя команда (3) начинает считывание блока данных с ленты в ячейки 2000–2099 до того, как был обработан предыдущий блок. Это называется досрочным чтением или *опережающим вводом* и делается в расчете на то, что данный блок в конце концов понадобится. Но на самом деле, начав обработку блока в ячейках 1000–1099, можно обнаружить, что никаких данных больше не нужно. Мы уже встречались с аналогичной ситуацией, например, в сопрограмме из раздела 1.4.2, где входные данные поступали с перфокарт, а не с ленты. Появление "." в любом месте перфокарты означало, что это последняя карта колоды. Подобная ситуация делает опережающий ввод невозможным. Исключение составляют случаи, когда можно предположить, что (a) за колодой перфокарт с входными данными следует пустая перфокарта или специальная дополнительная карта некоторого другого вида, либо (b), скажем, в колонке 80 последней карты колоды появляется опознавательная метка (например, "."). При использовании опережающего ввода для правильного завершения ввода в конце программы всегда должны быть предусмотрены специальные средства.

Метод параллельного выполнения вычислений и операций В/В называется *буферизацией*, в то время как элементарный метод (1) называется *небуферизированным* вводом. Область ячеек памяти 2000–2099, которая используется для сохранения опережающего ввода в (3), а также область ячеек 1000–1099, в которые перемещаются входные данные, называется *буфером*. В словаре Вебстера (Webster) *New World Dictionary* слово "буфер" определяется как "Любой человек или предмет, который служит для смягчения удара". Этот термин нам подходит, так как для буферизации характерна более ровная работа устройств В/В. (Инженеры-электронщики часто используют слово "буфер" в другом смысле, обозначая им часть устройства В/В, в которой сохраняется информация во время ее передачи. Но в этой книге

термин "буфер" будет означать область *памяти*, используемую программистом для хранения данных В/В.)

Последовательность (3) не всегда лучше, чем (1), хотя исключения из этого правила достаточно редки. Давайте сравним их время выполнения. Пусть T — время, необходимое для ввода 100 слов, и пусть C — время выполнения, которое проходит между запросами на ввод данных. Для метода (1) требуется, в основном, $T + C$ времени на блок ленты, а для метода (3) — в основном, $\max(C, T) + 202u$. (Величина $202u$ — это время, необходимое для выполнения двух команд MOVE.) Один из способов оценки времени выполнения состоит в том, чтобы рассмотреть время прохождения критического пути; в данном случае — промежуток времени между моментами использования устройства В/В, в течение которого оно не занято. В методе (1) устройство остается незанятым в течение C единиц времени, а в методе (3) — 202 единиц времени (в предположении, что $C < T$).

Относительно медленно работающие команды MOVE из (3) использовать нежелательно, особенно потому, что они отнимают время прохождения критического пути, когда накопитель на магнитной ленте должен быть неактивным. Но существует почти очевидный способ улучшения этого метода, который позволит избежать использования команд MOVE. Внешнюю программу можно переделать так, чтобы она обращалась поочередно к ячейкам 1000–1099 и 2000–2099. Во время считывания данных в один буфер можно выполнять вычисления в другом буфере; затем можно начать считывание в другой буфер, в то же время проводя вычисления над данными в первом буфере. Этот важный метод называется *свопингом*. Адрес текущего буфера сохраняется в индексном регистре (или, если нет свободных индексных регистров, в ячейке памяти). Мы уже встречались с методом свопинга, когда он применялся к выходным данным алгоритма 1.3.2P (см. шаги P9–P11) и к сопутствующей программе.

Рассмотрим пример применения метода свопинга ко вводу. Предположим, каждый блок ленты состоит из 100 отдельных элементов, содержащих по одному слову. Ниже приведена подпрограмма, которая берет следующее слово из входных данных и начинает считывание нового блока, если текущий исчерпан.

```
01  WORDIN STJ  1F          Сохранить адрес выхода.
02         INC6 1           Перейти к следующему слову.
03  2H     LDA  0,6         Это конец
04         CMPA =SENTINEL=      буфера?
05  1H     JNE  *           Если нет, выйти.
06         IN   -100,6(U)   Пополнить этот буфер.
07         LD6  1,6         Переключиться на другой
08         JMP  2B              буфер и вернуться.
09  INBUF1 ORIG *+100       Первый буфер.
10         CON  SENTINEL    Маркер конца буфера.
11         CON  *+1         Адрес другого буфера.
12  INBUF2 ORIG *+100       Второй буфер.
13         CON  SENTINEL    Маркер конца буфера.
14         CON  INBUF1      Адрес другого буфера.  ▮
```
(4)

В этой программе для адресации последнего слова ввода используется регистр 6. Предполагается, что вызывающая программа не изменяет его значение. Символ U

обозначает накопитель на магнитной ленте, а символ SENTINEL — это значение, о котором известно (из характеристик программы), что его *нет* ни в одном блоке на ленте.

По поводу данной подпрограммы необходимо отметить следующее.

1) Константа-маркер (sentinel)появляется в качестве 101-го слова каждого буфера и представляет собой удобное средство для обозначения окончания буфера. Но во многих задачах этот метод не будет надежным, так как на ленте может появиться любое слово. Если вводить данные с перфокарт, то подобный метод (когда 17-е слово буфера является маркером) всегда можно использовать, не опасаясь сбоя. В этом случае маркером может служить любое отрицательное число, так как при вводе в MIX с перфокарт всегда получаются неотрицательные слова.

2) В каждом буфере содержится адрес другого буфера (см. строки 07, 11 и 14). Эта "взаимосвязь" облегчает процесс свопинга.

3) Нет необходимости в команде JBUS, так как следующий ввод инициируется до того, как происходит обращение к какому-либо слову из предыдущего блока. Если величины C и T, как и раньше, обозначают время вычисления и время ввода данных с ленты, то время выполнения из расчета на блок ленты теперь составляет $\max(C, T)$. Поэтому, если $C \leq T$, ленту можно оставить работать на полной скорости. (*Примечание.* В данном отношении MIX — идеализированный компьютер, так как программа не должна обрабатывать никаких ошибок В/В. На большинстве машин непосредственно перед командой IN в этой программе необходимо было бы использовать команды проверки успешного завершения предыдущей операции.)

4) Чтобы подпрограмма (4) работала правильно, необходимо запустить еще кое-что, как только программа начнет работать. Мы предоставляем читателю самостоятельно разобраться в деталях (см. упр. 6).

5) Благодаря подпрограмме WORDIN для всей остальной программы дело выглядит таким образом, как будто блоки на магнитной ленте имеют длину, равную 1, а не 100. Такой способ разбиения одного блока на ленте на несколько программно-ориентированных записей называется *блокировкой записей.*

Методы, продемонстрированные здесь для ввода, с небольшими изменениями применимы и для вывода (см. упр. 2 и 3).

Множественная буферизация. Свопинг — это только частный случай при $N = 2$ общего метода для N буферов. В некоторых задачах желательно иметь более двух буферов. Для примера рассмотрим следующий тип алгоритма.

Шаг 1. Прочитать пять блоков один за другим.

Шаг 2. На основе этих данных выполнить достаточно трудоемкие вычисления.

Шаг 3. Вернуться к шагу 1.

В данном случае желательно иметь пять-шесть буферов, чтобы во время выполнения шага 2 можно было читать следующую порцию данных из пяти блоков. Благодаря этой тенденции передавать для В/В пакеты данных множественная буферизация получает большие преимущества по сравнению со свопингом.

Предположим, имеется N буферов для некоторого процесса ввода или вывода, выполняющегося с помощью единственного устройства В/В. Будем представлять себе, что эти буфера расположены по кругу, как показано на рис. 23. Можно считать,

что внешняя для процесса буферизации программа имеет следующий общий **вид** относительно нашего устройства В/В.

```
    ⋮
НАЗНАЧИТЬ
    ⋮
ОСВОБОДИТЬ
    ⋮
НАЗНАЧИТЬ
    ⋮
ОСВОБОДИТЬ
    ⋮
```

Другими словами, можно считать, что в программе чередуются действие под названием НАЗНАЧИТЬ и действие под названием ОСВОБОДИТЬ, между которыми выполняются другие вычисления, не оказывающие влияния на распределение буферов.

НАЗНАЧИТЬ означает, что программа получает адрес следующей буферной области; этот адрес присваивается как значение некоторой переменной, использующейся в программе.

ОСВОБОДИТЬ означает, что программа закончила работу с текущей буферной областью.

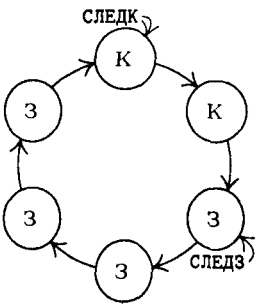


Рис. 23. Кольцо буферов ($N = 6$).

Между НАЗНАЧИТЬ и ОСВОБОДИТЬ программа работает с одним из буферов, который называется *текущей* буферной областью, а между ОСВОБОДИТЬ и НАЗНАЧИТЬ программа не обращается ни к одной буферной области.

Понятно, что НАЗНАЧИТЬ может непосредственно следовать за ОСВОБОДИТЬ. Рассмотрение вопросов буферизации часто основывалось на этом предположении. Но если ОСВОБОДИТЬ выполняется как можно быстрее, то процесс буферизации является более независимым и более эффективным. Разделяя эти две, по существу, разные функции, т. е. НАЗНАЧИТЬ и ОСВОБОДИТЬ, мы обнаружим, что метод буферизации остается легким для восприятия, а наше обсуждение — содержательным даже при $N = 1$.

Для большей определенности давайте рассмотрим операции ввода и вывода по отдельности. Для операции ввода будем предполагать, что мы имеем дело с устрой-

ством чтения перфокарт. Действие НАЗНАЧИТЬ означает, что программе нужна информация с новой перфокарты. Поэтому следует занести в индексный регистр адрес ячейки памяти, в которой находится образ следующей карты. Действие ОСВОБОДИТЬ выполняется, когда информация об образе текущей перфокарты больше не нужна, так как она некоторым способом обработана программой (возможно, скопирована в другую часть памяти и т. д.). Поэтому текущую буферную область теперь можно заполнить следующей порцией опережающего ввода.

Для операции вывода рассмотрим АЦПУ. Действие НАЗНАЧИТЬ происходит, когда нужна свободная буферная область, в которую помещается образ строки для печати. Мы хотим занести в индексный регистр адрес ячейки памяти, с которой начинается такая область. Действие ОСВОБОДИТЬ происходит, когда этот образ строки полностью помещен в буферную область в виде, готовом для печати.

Пример. Чтобы напечатать содержимое ячеек 0800–0823, можно записать следующее.

```
JMP  ASSIGNP    (Занести в rI5 адрес буфера.)
ENT1 0,5
MOVE 800(24)    Переместить 24 слова в выходной буфер.        (5)
JMP  RELEASEP
```

Здесь ASSIGNP и RELEASEP — это подпрограммы, выполняющие две описанные функции буферизации для АЦПУ.

В оптимальной ситуации с точки зрения компьютера для выполнения операции НАЗНАЧИТЬ времени практически не требуется. Для ввода это означает, что каждый образ перфокарты будет введен с опережением, так что данные уже имеются в наличии, когда программа готова их принять. А для вывода это означает, что в памяти всегда будет свободное место для записи образа строки. В любом случае время на ожидание устройств В/В не тратится.

Чтобы получше описать алгоритм буферизации и сделать его более красочным, будем говорить, что буферная область либо зеленая, либо желтая, либо красная (на рис. 24 они обозначены буквами З, Ж и К).

Зеленый цвет означает, что область готова для того, чтобы ее НАЗНАЧИЛИ, т. е. заполнена информацией с опережением (в случае ввода) либо является свободной (в случае вывода).

Желтый цвет означает, что область уже НАЗНАЧЕНА, но еще не СВОБОДНА, т. е. это текущий буфер, с которым работает программа.

Красный цвет означает, что область уже СВОБОДНА, т. е. это свободная область (в случае ввода) или область, заполненная информацией (в случае вывода).

На рис. 23 изображены два "указателя", направленных на обозначенные кружочками буферные области. Это, по сути, индексные регистры в программе. СЛЕДЗ и СЛЕДК расшифровываются как "следующий зеленый" и "следующий красный" буфер соответственно. Третий указатель, ТЕКУЩИЙ (рис. 24), указывает на желтый буфер, если он есть.

Приведенные ниже алгоритмы можно с одинаковым успехом применять как для ввода, так и для вывода, но для определенности сначала рассмотрим случай ввода с устройства чтения перфокарт. Предположим, что программа достигла состояния, показанного на рис. 23. Это означает, что четыре образа перфокарт были введены

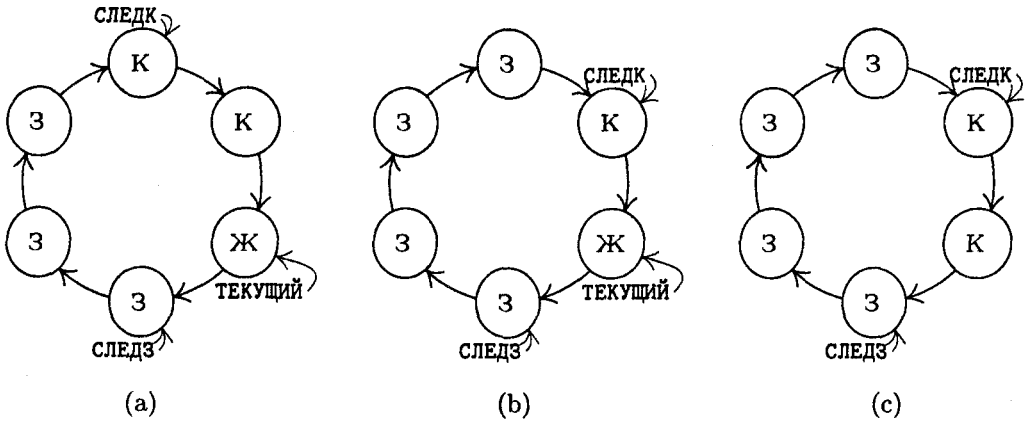


Рис. 24. Состояния буферов: (a) после НАЗНАЧИТЬ, (b) после завершения В/В и (c) после ОСВОБОДИТЬ.

с опережением с помощью процесса буферизации и находятся в зеленых буферах. В этот момент *одновременно* происходят два действия: (a) программа проводит вычисления по командам, следующим за ОСВОБОДИТЬ, и (b) перфокарта считывается в буфер, на который указывает СЛЕДК. Это состояние дел сохранится до тех пор, пока не завершится цикл ввода (и тогда устройство перейдет из состояния "занято" в состояние "готово") или пока программа не выполнит операцию НАЗНАЧИТЬ. Предположим, что сначала произошло последнее. Тогда буфер, на который указывает стрелка СЛЕДЗ, станет желтым (т. е. текущим), стрелка СЛЕДЗ сместится по часовой стрелке и получится конфигурация, изображенная на рис. 24, (a). Если к этому моменту ввод завершится, то останется еще один блок, введенный с опережением. Поэтому красный буфер станет зеленым, и стрелка СЛЕДК переместится, как показано на рис. 24, (b). Если следующей идет операция ОСВОБОДИТЬ, то получится конфигурация, изображенная на рис. 24, (c).

Пример, касающийся вывода, приведен на рис. 27 на с. 264. Здесь "цвета" буферных областей представлены как функция от времени. При этом рассматривается программа, которая начинает работу с четырех быстрых выводов, затем медленно выдает еще четыре вывода и в конце концов выдает два вывода подряд. В этом примере используются три буфера.

Указатели СЛЕДК и СЛЕДЗ весело продвигаются по кругу по часовой стрелке, каждый со своей скоростью. Это состязание между программой (которая делает зеленые буфера красными) и процессами буферизации В/В (которые делают красные буфера зелеными). В подобном случае могут возникнуть две конфликтные ситуации.

a) Если СЛЕДЗ пытается обогнать СЛЕДК, то программа опережает устройство В/В и вынуждена ждать, пока оно будет готово.
b) Если СЛЕДК пытается обогнать СЛЕДЗ, то устройство В/В опережает программу и необходимо остановить его до тех пор, пока не будет выполнена следующая операция RELEASE.

Обе эти ситуации отображены на рис. 27 (см. упр. 9).

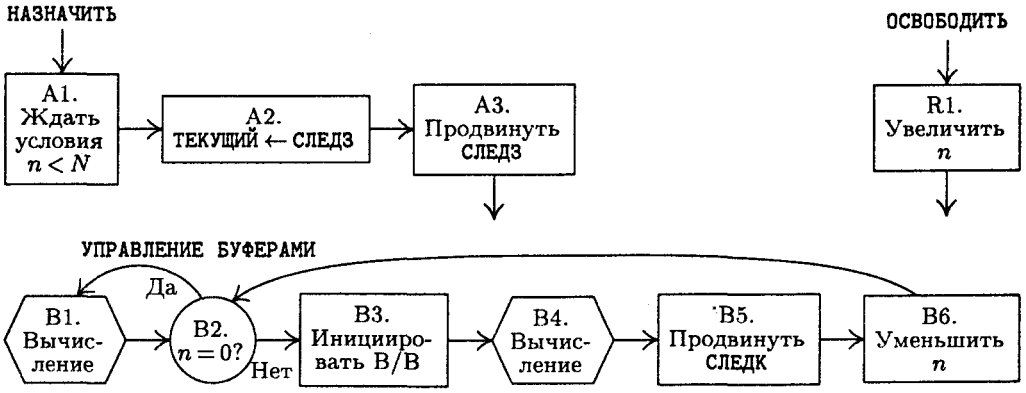


Рис. 25. Алгоритмы множественной буферизации.

К счастью, несмотря на довольно длинное объяснение, которое только что было приведено по поводу кольца буферов, сами алгоритмы обработки данной ситуации довольно просты. В следующем описании будем использовать такие обозначения:

$$\begin{aligned} N &\text{ — общее число буферов;} \\ n &\text{ — текущее число красных буферов.} \end{aligned} \tag{6}$$

В приведенном ниже алгоритме переменная n используется для того, чтобы СЛЕДЗ и СЛЕДК не мешали один другому.

Алгоритм A (НАЗНАЧИТЬ). Этот алгоритм (рис. 25) состоит из шагов, которые определяются операцией НАЗНАЧИТЬ в вычислительной программе, как описано выше.

A1. [Ждать условия $n < N$.] Если $n = N$, остановить программу, пока не будет выполнено условие $n < N$. (Если $n = N$, следовательно, ни один буфер не готов к тому, чтобы быть назначенным. Но приведенный ниже алгоритм B, работающий параллельно с данным алгоритмом в конце концов достигнет успеха в получении зеленого буфера.)

A2. [ТЕКУЩИЙ ← СЛЕДЗ.] Присвоить CURRENT ← NEXTG (назначая, таким образом, текущий буфер).

A3. [Продвинуть СЛЕДЗ.] Продвинуть СЛЕДЗ к следующему буферу по часовой стрелке. ▮

Алгоритм R (ОСВОБОДИТЬ). Этот алгоритм состоит из шагов, которые определяются операцией ОСВОБОДИТЬ в вычислительной программе, как описано выше.

R1. [Увеличить n.] Увеличить n на единицу. ▮

Алгоритм B (*Управление буферами*). Этот алгоритм осуществляет реальную инициацию операторов В/В в машине; он должен выполняться одновременно с главной программой, в том смысле, как описано ниже.

B1. [Вычисление.] Позволить главной программе в течение короткого времени проводить вычисления. Шаг B2 выполняется после некоторой задержки, когда устройство В/В готово для следующей операции.

B2. [$n = 0$?] Если $n = 0$, перейти к B1. (Таким образом, если нет ни одного красного буфера, нельзя выполнить ни одну операцию В/В.)

B3. [Инициировать В/В.] Инициировать передачу данных между буферной областью, на которую указывает СЛЕДК, и устройством В/В.

B4. [Вычисление.] Позволить главной программе работать в течение некоторого времени; затем, когда операция В/В будет завершена, перейти к шагу B5.

B5. [Продвинуть СЛЕДК.] Продвинуть СЛЕДК к следующему буферу по часовой стрелке.

B6. [Уменьшить n.] Уменьшить n на единицу и перейти к шагу B2. ▮

В этих алгоритмах присутствуют два независимых процесса, выполняющихся "одновременно", — программа управления буферизацией и вычислительная программа. Фактически эти процессы являются *сопрограммами*, которые мы назовем CONTROL и COMPUTE*. Сопрограмма CONTROL вызывает COMPUTE на шагах B1 и B4; сопрограмма COMPUTE вызывает CONTROL с помощью команд "перехода при готовности", которые разбросаны в программе через случайные промежутки.

Запрограммировать этот алгоритм для MIX чрезвычайно просто. Для удобства будем считать, что буфера связаны таким образом, что слово, *предшествующее* каждому из них, — это адрес следующего буфера. Например, при $N = 3$ имеем CONTENTS(BUF1 − 1) = BUF2, CONTENTS(BUF2 − 1) = BUF3 и CONTENTS(BUF3 − 1) = BUF1.

Программа A (НАЗНАЧИТЬ; *подпрограмма в сопрограмме* COMPUTE). rI4 ≡ ТЕКУЩИЙ; rI6 ≡ n; последовательность вызова: JMP ASSIGN; на выходе в rX содержится СЛЕДЗ.

```
НАЗНАЧИТЬ STJ  9F
1H        JRED CONTROL(U)  A1. Ожидать условия n < N.
          CMP6 =N=
          JE   1B
          LD4  СЛЕДЗ       A2. ТЕКУЩИЙ ← СЛЕДЗ.
          LDX  -1,4        A3. Продвинуть СЛЕДЗ.
          STX  СЛЕДЗ
9H        JMP  *           Выход. ▮
```

Программа R (ОСВОБОДИТЬ; *используется в сопрограмме* COMPUTE). rI6 ≡ n. Этот маленький фрагмент следует вставлять везде, где нужно выполнить операцию RELEASE.

```
          INC6 1           R1. Увеличить n.
          JRED CONTROL(U)  Возможен переход к сопрограмме CONTROL. ▮
```

Программа B (*Сопрограмма* CONTROL). rI6 ≡ n, rI5 ≡ СЛЕДК.

```
CONT1     JMP  COMPUTE     B1. Вычисление.
1H        J6Z  *-1         B2. n = 0?
          IN   0,5(U)      B3. Инициация В/В.
          JMP  COMPUTE     B4. Вычисление.
          LD5  -1,5        B5. Продвинуть СЛЕДК.
          DEC6 1           B6. Уменьшить n.
          JMP  1B ▮
```

* "Управление" и "вычисление". — *Прим. перев.*

Помимо приведенного выше кода, имеем также обычные команды связи сопрограмм:

```
CONTROL   STJ  COMPUTEX      COMPUTE   STJ  CONTROLX
CONTROLX  JMP  CONT1        COMPUTEX  JMP  COMP1
```

Кроме того, команду "JRED CONTROL(U)" следует помещать внутри программы COMPUTE примерно через каждые 50 команд.

Таким образом, программы множественной буферизации состоят, в сущности, только из семи команд для CONTROL, восьми — для НАЗНАЧИТЬ и двух — для ОСВОБОДИТЬ.

Здесь есть один замечательный факт: *один и тот же* алгоритм можно применять и для ввода, и для вывода. Но в чем же отличие между этими двумя процессами? Как программа управления определяет, что нужно делать — опережать (в случае ввода) или запаздывать (в случае вывода)? Ответ на этот вопрос кроется в начальных условиях. При вводе мы начинаем с $n = N$ (все буфера красные), а при выводе — с $n = 0$ (все буфера зеленые). И после того как программа запустится при соответствующих начальных условиях, она уже будет себя вести либо как процесс ввода, либо как процесс вывода. Другим начальным условием является NEXTR = NEXTG, т. е. обе стрелки должны указывать на один из буферов.

Для завершения программы необходимо остановить процесс ввода-вывода (если это ввод) или подождать, пока он закончится (если это вывод); подробности мы оставляем читателю (см. упр. 12 и 13).

Здесь возникает важный вопрос: "Какое значение N является оптимальным?". Конечно, по мере увеличения N скорость работы программы не уменьшится, но и не будет неограниченно расти, поэтому рано или поздно скорость роста снизится. Давайте снова обратимся к величинам C и T, которые представляют время вычислений между операторами В/В и само время В/В. Точнее, пусть C — это промежуток времени между последовательными операциями НАЗНАЧИТЬ, а T — промежуток времени, за которое передается один блок. Если C всегда *больше* T, то вполне достаточно будет значения $N = 2$, так как нетрудно показать, что с двумя буферами компьютер будет постоянно занят. Если C всегда *меньше* T, то и в такой ситуации достаточно значения $N = 2$, поскольку устройство В/В будет постоянно занято (за исключением случая, когда устройство имеет особые ограничения на время использования, как в упр. 19). Следовательно, бóльшие значения N полезны, главным образом, тогда, когда C принимает то малые, то большие значения. При этом, если большие значения C существенно превосходят T, подходящим значением N является среднее число последовательных малых величин C плюс 1. (Нужно заметить, однако, что преимущества буферизации фактически сводятся на нет, если ввод полностью выполняется в начале программы, а вывод — полностью в конце.) Если промежуток времени между операциями НАЗНАЧИТЬ и ОСВОБОДИТЬ всегда достаточно мал, то во всех описанных выше случаях значение N можно уменьшить на 1, что почти не окажет влияния на время выполнения.

Этот подход к буферизации можно изменять различными способами, и мы кратко расскажем о некоторых из них. До сих пор предполагалось, что используется только одно устройство В/В, но, конечно, на практике вы будете применять несколько устройств одновременно.

Существует несколько подходов к ситуации, когда используется несколько устройств. В простейшем случае у нас будет отдельное кольцо буферов для каждого устройства. Для каждого устройства будут заданы свои значения n, N, СЛЕДК, СЛЕДЗ и ТЕКУЩИЙ, а также собственная сопрограмма CONTROL. В результате мы будем иметь эффективную буферизацию одновременно на всех устройствах В/В.

Можно также создать "пул" буферных областей одинакового размера, чтобы два или более устройств совместно использовали буфера из общего списка. Это можно осуществить с помощью методов связывания памяти, которые описываются в главе 2, когда все красные буфера ввода связываются в один список, а все зеленые буфера вывода — в другой. В данном случае возникает необходимость отличать ввод от вывода, т. е. нужно переписать алгоритмы, чтобы в них не использовались n и N. Если все буфера в пуле окажутся заполненными входными данными, полученными методом опережающего ввода, то алгоритм будет неизменно останавливаться. Поэтому необходимо убедиться, что существует по крайней мере один буфер (причем желательно по одному для каждого устройства), который не является зеленым буфером ввода. И только если программа COMPUTE останавливается на шаге А1 для некоторого устройства ввода, следует разрешить ввод данных с этого устройства в последний буфер пула.

Некоторые машины имеют дополнительные ограничения на использование устройств ввода-вывода, делающие невозможной одновременную передачу данных с определенных пар устройств. (Например, несколько устройств можно подключить к компьютеру с помощью единственного "канала".) Это ограничение также оказывает влияние на нашу программу буферизации. Как сделать правильный выбор, когда нужно решить, какое устройство В/В инициировать следующим? Это называется задачей прогнозирования. Лучшее правило прогнозирования для общего случая, видимо, состоит в том, чтобы отдать предпочтение устройству, буферное кольцо которого имеет наибольшее значение n/N. При этом предполагается, что число буферов в кольце было выбрано обдуманно.

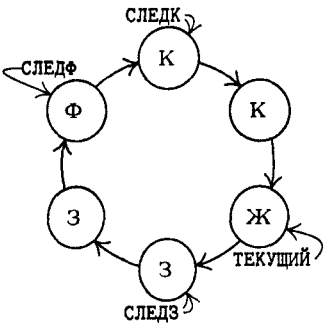


Рис. 26. Ввод и вывод в одном и том же кольце буферов.

И в заключение рассмотрим полезный метод выполнения и ввода, и вывода с помощью *одного и того же* кольца буферов, который можно применять при определенных условиях. На рис. 26 показан новый вид буфера — фиолетовый. В этой ситуации зеленые буфера представляют опережающий *ввод*; в результате операции НАЗНАЧИТЬ зеленый буфер становится желтым, а затем, после операции ОСВОБОДИТЬ, — красным и представляет блок, который нужно *вывести*. Процессы ввода и вывода, как и раньше, независимо двигаются по кругу, только сейчас после выполнения операции вывода красные буфера становятся фиолетовыми, а при

вводе фиолетовый буфер превращается в зеленый. Необходимо гарантировать, что указатели СЛЕДЗ, СЛЕДК и СЛЕДФ не будут обгонять друг друга. В момент, показанный на рис. 26, программа выполняет вычисления между операциями НАЗНАЧИТЬ и ОСВОБОДИТЬ, работая с желтым буфером; одновременно производится ввод в буфер, на который указывает стрелка СЛЕДФ, и вывод из буфера, на который указывает стрелка СЛЕДК.

УПРАЖНЕНИЯ

1. [*05*] *Будет ли последовательность команд* (3) *по-прежнему правильной, если поместить* команды MOVE перед командой JBUS, а не после нее? А если поместить команды MOVE после команды IN?

2. [*10*] С помощью команд "OUT 1000(6); JBUS *(6)" можно вывести блок данных на ленту, не используя буферизацию, точно так же, как команды (1) делают это в случае ввода. Предложите метод, аналогичный (2) и (3), который буферизирует этот вывод, используя команды MOVE и вспомогательный буфер, занимающий ячейки 2000–2099.

▸ **3.** [*22*] Напишите подпрограмму вывода с помощью свопинга, аналогичную (4). Эта подпрограмма с именем WORDOUT должна сохранять слово в rA в качестве следующего слова вывода, а когда буфер будет заполнен, записывать 100 слов на магнитную ленту (устройство V). В индексном регистре 5 должен храниться адрес текущего буфера. Составьте схему расположения буферных областей и объясните, какие команды необходимо (если необходимо вообще) использовать в начале и в конце программы, чтобы первый и последний блоки наверняка были записаны правильно. В случае необходимости последний блок следует заполнить нулями.

4. [*M20*] Покажите, что если программа использует единственное устройство В/В, то при благоприятных обстоятельствах можно сократить время ее выполнения наполовину путем буферизации В/В. Но нельзя более чем в два раза сократить время выполнения по сравнению с небуферизированным В/В.

▸ **5.** [*M21*] Обобщите решение предыдущего упражнения для случая, когда программа работает не с одним, а с n устройствами В/В.

6. [*12*] Какие команды нужно поместить в начале программы, чтобы подпрограмма WORDIN (4) начала правильно работать? (Например, в индексном регистре 6 должно *что-то* содержаться.)

7. [*22*] Напишите подпрограмму с именем WORDIN, которая, в основном, аналогична (4), за исключением того, что в ней не используется маркер конца блока.

8. [*11*] В тексте раздела описывается гипотетический сценарий ввода, начало которого показано на рис. 23, а продолжение — на рис. 24, (a), (b) и (c). Дайте интерпретацию такому же сценарию, но при условии, что выполняется вывод на АЦПУ, а не ввод с перфокарт. (Например, что происходит в момент, показанный на рис. 23?)

▸ **9.** [*21*] Программу, в результате выполнения которой содержимое буферов выглядит, как показано на рис. 27, можно охарактеризовать с помощью следующего списка промежутков времени:

$$A, 1000, R, 1000, A, 1000, R, 1000, A, 1000, R, 1000, A, 1000, R, 1000, A, 1000, R, 1000,$$
$$A, 7000, R, 5000, A, 7000, R, 5000, A, 7000, R, 5000, A, 7000, R, 5000,$$
$$A, 1000, R, 1000, A, 2000, R, 1000.$$

Этот список расшифровывается так: "Назначить, вычислять в течение $1000u$, освободить, вычислять в течение $1000u$, назначить, ..., вычислять в течение $2000u$, освободить, вычислять в течение $1000u$". Приведенные промежутки времени вычислений не включают

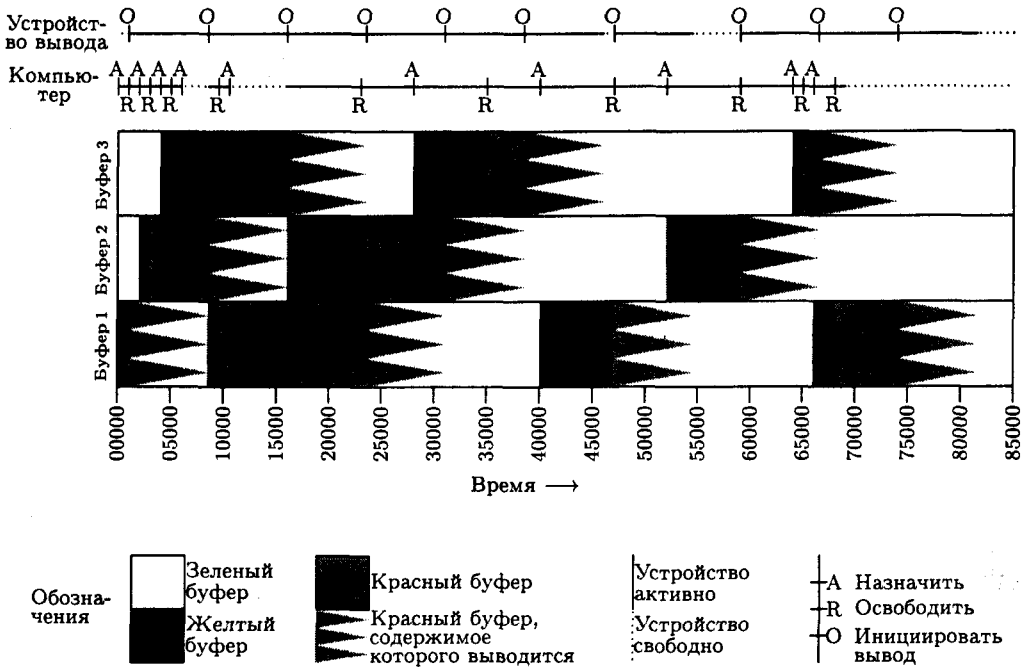


Рис. 27. Вывод с использованием трех буферов (см. упр. 9).

какие-либо периоды, когда компьютер ожидает устройство вывода, чтобы занять его (как в случае четвертой операции "назначить" на рис. 27). Выходное устройство затрачивает на вывод каждого блока по $7500u$ времени.

В следующей таблице перечислены действия, которые соответствуют промежуткам времени, показанным на рис. 27.

Время	Действие	Время	Действие
0	ASSIGN(BUF1)	38500	OUT BUF3
1000	RELEASE, OUT BUF1	40000	ASSIGN(BUF1)
2000	ASSIGN(BUF2)	46000	Вывод остановлен.
3000	RELEASE	47000	RELEASE, OUT BUF1
4000	ASSIGN(BUF3)	52000	ASSIGN(BUF2)
5000	RELEASE	54500	Вывод остановлен.
6000	ASSIGN (ждать)	59000	RELEASE, OUT BUF2
8500	BUF1 назначен, OUT BUF2	64000	ASSIGN(BUF3)
9500	RELEASE	65000	RELEASE
10500	ASSIGN (ждать)	66000	ASSIGN(BUF1)
16000	BUF2 назначен, OUT BUF3	66500	OUT BUF3
23000	RELEASE	68000	RELEASE
23500	OUT BUF1	69000	Вычисления прекращены.
28000	ASSIGN(BUF3)	74000	OUT BUF1
31000	OUT BUF2	81500	Вывод остановлен.
35000	RELEASE		

Таким образом, всего потребовалось $81500u$; компьютер простаивал в промежутках 6000–8500, 10500–16000 и 69000–81500, т. е. всего $20500u$; устройство вывода было свободно в промежутках 0–1000, 46000–47000 и 54500–59000, т. е. всего $6500u$.

Для этой же программы создайте таблицу типа "время — действие", аналогичную приведенной выше, но при условии, что используются только *два* буфера.

10. [*21*] Выполните упр. 9, но только для *четырех* буферов.

11. [*21*] Выполните упр. 9, но только для *одного* буфера.

12. [*24*] Предположим, что алгоритм множественной буферизации, приведенный в тексте, используется для ввода с перфокарт. И пусть ввод должен быть прекращен сразу же, как только будет считана перфокарта, в колонке 80 которой содержится ".". Покажите, как следует модифицировать сопрограмму CONTROL (алгоритм B и программу B), чтобы ввод прекращался указанным образом.

13. [*20*] Пусть вывод выполняется с помощью алгоритмов буферизации. Какие команды нужно вставить в конце сопрограммы COMPUTE, приведенной в тексте раздела, чтобы гарантировать вывод всей информации из буферов?

▸ **14.** [*20*] Предположим, в вычислительной программе нет чередования действий НАЗНАЧИТЬ и ОСВОБОДИТЬ, а есть только последовательность действий ... НАЗНАЧИТЬ ... НАЗНАЧИТЬ ... ОСВОБОДИТЬ ... ОСВОБОДИТЬ. Какое влияние это окажет на алгоритмы, описанные в тексте раздела? Может ли это оказаться полезным?

▸ **15.** [*22*] Напишите законченную программу для MIX, которая копирует 100 блоков с накопителя на магнитной ленте под номером 0 на аналогичное устройство номер 1, используя только три буфера. Программа должна работать настолько быстро, насколько это возможно.

16. [*29*] Сформулируйте алгоритм для зеленого-желтого-красного-фиолетового буферов, которые предложены на рис. 26, аналогичный алгоритмам множественной буферизации, приведенным в тексте раздела. Используйте три сопрограммы (одну для управления устройством ввода, другую — устройством вывода и третью — для вычислений).

17. [*40*] Переделайте алгоритм множественной буферизации для пула буферов; предусмотрите встроенные методы, которые не допускают замедления процесса из-за слишком большого объема опережающего ввода. Постарайтесь, по возможности, придать алгоритму красоту и изящество. Сравните свой метод с методами, в которых не используется пул, применяя их к реальным задачам.

▸ **18.** [*30*] Предлагаемое расширение машины MIX позволяет прерывать вычисления, как будет описано ниже. Ваша задача в этом упражнении — модифицировать приведенные в тексте раздела алгоритмы и программы A, R и B, чтобы вместо команд JRED в них использовались эти средства прерывания.

Новые возможности MIX включают 3 999 дополнительных ячеек памяти с адресами от −3999 до −0001. У этой машины есть два внутренних "состояния" — *нормальное* и *управляющее*. В нормальном состоянии ячейки с −3999 по −0001 недоступны и машина MIX работает, как обычно. Когда происходит "прерывание", вызванное условиями, о которых речь пойдет позже, в ячейки с −0009 по −0001 заносится содержимое регистров машины MIX: rA — в −0009; rI1–rI6 — в −0008–−0003; rX — в −0002, а rJ, состояние флага переполнения, флага сравнения и адрес следующей команды сохраняются в ячейке −0001 в следующем виде:

+	след. ком..	OV, CI	rJ

Когда машина входит в управляющее состояние, ячейка, которой передается управление, выбирается в зависимости от типа прерывания.

Ячейка −0010 играет роль часов: через каждые $1000u$ единиц времени число, содержащееся в этой ячейке, уменьшается на единицу; если в результате получается нуль, то происходит прерывание и управление передается в ячейку −0011.

Новая команда MIX "INT" (C = 5, F = 9) работает следующим образом. (a) В нормальном состоянии при прерывании управление передается ячейке −0012. (Таким образом, программист может вызвать прерывание, чтобы установить связь с управляющей программой; адрес INT значения не имеет, хотя управляющая программа может использовать его в информационном смысле, чтобы отличать один тип прерывания от другого.) (b) В управляющем состоянии все регистры MIX загружаются информацией из ячеек с −0009 по −0001, затем компьютер возвращается в нормальное состояние и возобновляет выполнение. Время выполнения команды INT в обоих случаях составляет $2u$.

Команда IN, OUT или IOC, выданная в *управляющем* состоянии, вызовет прерывание сразу же по окончании операции В/В. В этом случае управление будет передано в ячейку −(0020+ номер устройства).

В управляющем состоянии прерывания никогда не происходят. Любые условия, вызывающие прерывания, "сохраняются" до появления следующей операции INT, и прерывание происходит после выполнения одной команды в нормальном состоянии программы.

▸ **19.** [*M28*] Ввод-вывод небольших блоков данных с вращающегося устройства, например с магнитного диска, необходимо рассматривать особо. Предположим, программа работает с $n \geq 2$ последовательными блоками информации следующим образом. Блок k начинает вводиться в момент t_k, где $t_1 = 0$. Он назначается для обработки в момент $u_k \geq t_k + T$ и освобождает буфер в момент $v_k = u_k + C$. Диск совершает один оборот через каждые P единиц времени, и его считывающая головка пересекает начало нового блока через каждые L единиц времени; поэтому мы имеем $t_k \equiv (k-1)L$ (по модулю P). Так как обработка выполняется последовательно, имеем также $u_k \geq v_{k-1}$ при $1 < k \leq n$. Всего у нас N буферов; следовательно, $t_k \geq v_{k-N}$ при $N < k \leq n$.

Насколько большим должно быть N, чтобы время v_n завершения операции было минимальным из возможных, $T + C + (n-1)\max(L, C)$? Сформулируйте общее правило определения наименьшего такого N. Проиллюстрируйте свое правило для $L = 1$, $P = 100$, $T = .5$, $n = 100$ и (a) $C = .5$; (b) $C = 1.0$; (c) $C = 1.01$; (d) $C = 1.5$; (e) $C = 2.0$; (f) $C = 2.5$; (g) $C = 10.0$; (h) $C = 50.0$; (i) $C = 200.0$.

Поэтому в нашем примере имеем (a) $N = 1$; (b) $N = 2$; (c) $N = 3$, $c_N = 2.5$; (d) $N = 35$, $c_N = 51.5$; (e) $N = 51$, $c_N = 101.5$; (f) $N = 41$, $c_N = 102$; (g) $N = 11$, $c_N = 109.5$; (h) $N = 3$, $c_N = 149.5$; (i) $N = 2$, $c_N = 298.5$.

1.4.5. История и библиография

Большинство фундаментальных методов, описанных в разделе 1.4, было независимо разработано разными программистами, и подробная история возникновения их идей, видимо, никогда не будет достоверно известна. Поэтому сейчас мы просто попытаемся перечислить и оценить наиболее важные работы, которые внесли вклад в развитие этих идей.

Подпрограммы были первым средством экономии сил программистов. Еще в 19 веке Чарльз Бэббидж (Charles Babbage) предвидел возможность создания библиотеки программ для своей аналитической машины [см. *Charles Babbage and His Calculating Engines*, под редакцией Филиппа (Philip) и Эмили (Emily) Моррисон (Morrison) (Dover, 1961), 56]. И теперь можно сказать, что его мечта сбылась в 1944 году, когда Грэйс М. Хоппер (Grace M. Hopper) написала подпрограмму вычисления функции $\sin x$ для счетно-решающего устройства Mark I, установленного

в Гарвардском университете [см. *Mechanization of Thought Processes* (London: Nat. Phys. Lab., 1959), 164]. Но это еще были, в сущности, "открытые подпрограммы", т. е. такие подпрограммы, которые нужно вставлять в программу, а не связывать динамически. Управление машиной Бэббиджа, как и ткацким станком Жаккарда, осуществлялось с помощью набора перфокарт, а счетно-решающим устройством Mark I — с помощью бумажных перфолент. Таким образом, эти устройства существенно отличались от современных компьютеров с их хранящимися в памяти программами.

Связь с подпрограммами, реализуемая на машинах с хранящимися в памяти программами, где адрес, по которому нужно вернуть управление, передается в качестве параметра, была рассмотрена Германом Г. Голдстайном (Herman H. Goldstine) и Джоном фон Нейманом (John von Neumann) в их широкоизвестной монографии по программированию, написанной между 1946 и 1947 годами [см. работу фон Неймана *Collected Works* **5** (New York: Macmillan, 1963), 215–235]. В их программах главная программа отвечала за передачу параметров в тело подпрограммы, а не за передачу необходимой информации в регистры. В Англии А. М. Тьюринг (A. M. Turing) уже в 1945 году разработал аппаратное и программное обеспечение связи с подпрограммами [см. работу *Proceedings of a Second Symposium on Large-Scale Digital Calculating Machinery* (Cambridge, Mass.: Harvard University, 1949), 87–90; B. E. Carpenter, R. W. Doran, editors, *A. M. Turing's ACE Report of 1946 and Other Papers* (Cambridge, Mass.: MIT Press, 1986), 35, 36, 76, 78, 79]. Способы применения и строение универсальной библиотеки подпрограмм — это главная тема первого учебника по компьютерному программированию M. V. Wilkes, D. J. Wheeler, S. Gill, *The Preparation of Programs for an Electronic Digital Computer*, 1st ed. (Reading, Mass.: Addison-Wesley, 1951) (Уилкс М., Уилер Д. и Гилл С. Составление программ для электронных счетных машин. — М.: Изд-во иностр. лит., 1953).

Слово "сопрограмма" придумал М. Э. Конвей (M. E. Conway) в 1958 году, после того как он разработал это понятие и впервые применил его при построении программы на языке ассемблера. Почти в то же время сопрограммы независимо изучали Дж. Эрдвинн (J. Erdwinn) и Дж. Мернер (J. Merner). Они написали статью "Bilateral Linkage"*, которую посчитали недостаточно интересной, чтобы быть достойной опубликования. К сожалению, до сегодняшнего дня ни один экземпляр этой статьи, похоже, не сохранился. Первое опубликованное объяснение понятия сопрограммы появилось значительно позже в статье Конвея "Design of a Separable Transition-Diagram Compiler", *CACM* **6** (1963), 396–408. На самом деле о примитивной форме связи сопрограмм уже кратко упоминалось в "советах по программированию" в первых публикациях, посвященных машине UNIVAC [*The Programmer* **1**, 2 (February, 1954), 4]. Удобное обозначение для сопрограмм в алголоподобных языках было введено в работе Dahl, Nygaard SIMULA I [*CACM* **9** (1966), 671–678]. Несколько прекрасных примеров сопрограмм (включая сопрограммы репликации) появилось в книге O.-J. Dahl, E. W. Dijkstra, C. A. R. Hoare *Structured Programming*, Chapter 3 (Дал У., Дейкстра Э., Хоор К. Структурное программирование / Пер. с англ. — М.: Мир, 1975 (сер. "Математическое обеспечение ЭВМ")). Первой программой-интерпретатором можно считать "универсальную машину Тьюринга", способную имитировать любые другие машины Тьюринга. Машины Тьюринга —

* "Двусторонняя связь". — *Прим. перев.*

это не реальные компьютеры, а теоретические конструкции, использующиеся для доказательства того, что некоторые задачи нельзя решить с помощью алгоритмов. Программы-интерпретаторы в традиционном смысле упоминал в 1946 году Джон Мочли (John Mauchly) в своих лекциях, которые он читал в Школе Мура (Moore School). Самыми известными из первых интерпретаторов, главным назначением которых было обеспечение удобных средств выполнения операций с плавающей точкой, были программы для машины Whirlwind I (разработанные Ч. В. Адамсом (C. W. Adams) и др.) и для машины Illiac I (разработанные Д. Дж. Уилером (D. J. Wheeler) и др.). Тьюринг также принимал участие в их разработке; интерпретирующие системы для компьютера Pilot ACE были написаны под его руководством. Более подробно о положении дел с интерпретаторами в начале 50-х годов говорится в статье J. M. Bennett, D. G. Prinz, M. L. Woods, "Interpretative Sub-routines", *Proc. ACM Nat. Conf.* (1952), 81–87 [см. также различные статьи в журнале *Proceedings of the Symposium on Automatic Programming for Digital Computers* (1954), издаваемом Департаментом морских исследований (Вашингтон, федеральный округ Колумбия).

Наиболее широкоиспользуемой программой-интерпретатором, вероятно, была "IBM 701 Speedcoding system" Джона Бэкуса (John Backus) [см. *JACM* **1** (1954), 4–6]. Этот интерпретатор был несколько модифицирован и искусно переписан для IBM 650 В. М. Волонтисом (V. M. Wolontis) и другими из компании Bell Telephone Laboratories; их программа "Bell Interpretive System" приобрела широкую популярность. Интерпретирующие системы IPL, которые были разработаны в начале 1956 года А. Ньювеллом (A. Newell), Дж. К. Шоу (J. C. Shaw) и Г. А. Саймоном (H. A. Simon) для совершенно других целей (см. раздел 2.6), широко использовались для обработки списков. Как уже упоминалось во введении к разделу 1.4.3, о современных способах применения интерпретаторов в компьютерной литературе, как правило, говорится "на ходу" [см. ссылки на статьи, перечисленные в этом разделе, в которых интерпретаторы обсуждаются более подробно].

Первая программа трассировки была разработана Стэнли Гиллом (Stanley Gill) в 1950 году [см. его интересную статью в журнале *Proceedings of the Royal Society of London*, series A, **206** (1951), 538–554]. В упомянутой выше книге Уилкса, Уилера и Гилла содержится несколько программ трассировки. Наверное, самая интересная из них — это подпрограмма С-10, написанная Д. Дж. Уилером, в которой предусмотрена возможность, позволяющая приостановить трассировку при входе в библиотечную подпрограмму, выполнить эту подпрограмму на полной скорости, а затем продолжить трассировку. Информация о программах трассировки довольно редко публикуется в общей компьютерной литературе. Это обусловлено, главным образом, тем, что данные методы по своей природе ориентированы на конкретную машину. Автору известна только одна ранняя работа на эту тему — статья H. V. Meek, "An Experimental Monitoring Routine for the IBM 705", *Proc. Western Joint Computer Conf.* (1956), 68–70. В ней обсуждается программа трассировки для машины, на которой было чрезвычайно трудно решить одну задачу. В настоящее время внимание к программам трассировки сместилось в сторону программ, обеспечивающих выборочный вывод результатов и оценок производительности программы; одна из лучших подобных систем была разработана Э. Сэттерсвейтом (E. Satterthwaite) и описана в журнале *Software Practice & Experience* **2** (1972), 197–217.

Буферизация первоначально выполнялась аппаратным способом, аналогично тому, как это делалось в программе 1.4.4–(3). Внутренняя буферная область, недо-

ступная для программиста, играла роль ячеек 2000–2099, и команды 1.4.4–(3) выполнялись неявно и незаметно, когда выдавалась команда ввода. В конце 40-х годов первые программисты UNIVAC разработали программные методы буферизации, которые особенно полезны для сортировки (см. раздел 5.5). Хороший обзор основных принципов В/В, преобладавших в 1952 году, можно найти в Трудах конференции Восточного компьютерного объединения, которая проводилась в 1952 году.

В компьютере DYSEAC [Alan L. Leiner, *JACM* **1** (1954), 57–81] была реализована идея непосредственной передачи информации между устройствами ввода-вывода и памятью во время работы программы, а затем прерывания программы по ее завершении. Из существования подобных систем следует, что для них были разработаны алгоритмы буферизации, но подробности не были опубликованы. В первой опубликованной работе о методах буферизации в том смысле, в котором мы их описали, представлен крайне сложный подход к этой проблеме [см. O. Mock, C. J. Swift, "Programmed Input-Output Buffering", *Proc. ACM Nat. Conf.* (1958), paper 19, и *JACM* **6** (1959), 145–151]. (Должен предупредить читателя о том, что в обеих статьях широко используется местный жаргон, для понимания которого придется потратить некоторое время, но вам помогут в этом другие статьи из журнала *JACM* **6**.) Система прерывания, которая позволяла осуществлять буферизацию ввода и вывода, была независимо разработана Э. В. Дейкстрой (E. W. Dijkstra) в 1957 и 1958 годах для компьютера $X-1$, созданного Б. Ж. Лупстрой (B. J. Loopstra) и К. С. Шолтеном (C. S. Scholten) [см. *Comp. J.* **2** (1959), 39–43]. В докторской диссертации Дейкстры, "Communication with an Automatic Computer"* (1958), обсуждаются методы буферизации, которые в данном случае были рассчитаны на очень длинные кольца буферов, в то время как в программах рассматривался, в основном, В/В на перфоленты и терминал. В каждом буфере содержался либо один символ, либо одно число. Впоследствии Дейкстра развил эти идеи и пришел к важному общему понятию *семафоров*, которые лежат в основе управления параллельными процессами любого вида, а не только вводом-выводом [см. *Programming Languages*, ed. by F. Genuys (Academic Press, 1968), 43–112; *BIT* **8** (1968), 174–186; *Acta Informatica* **1** (1971), 115–138]. В статье David E. Ferguson, "Input-Output Buffering and FORTRAN", *JACM* **7** (1960), 1–9, рассматриваются буферные кольца и приводится подробное описание простой буферизации для многих устройств одновременно.

Как представляется, примерно 1 000 команд — это разумный верхний предел сложности задач.

— ГЕРМАН ГОЛДСТАЙН (HERMAN GOLDSTINE) и ДЖОН ФОН НЕЙМАН (JOHN VON NEUMANN) (1946)

---

* "Связь с помощью автоматического компьютера". — *Прим. перев.*

ГЛАВА 2

ИНФОРМАЦИОННЫЕ СТРУКТУРЫ

Нет, не увижу, это ясно,
Поэмы, дерева прекрасней.

— ДЖОЙС КИЛМЕР (JOICE KILMER) (1913) (Перевод Л. Макаровой)

Ах, я с таблицы памяти моей
Все суетные записи сотру.

— *Гамлет* (акт I, сцена 5, строка 98) (Перевод М. Лозинского)

2.1. ВВЕДЕНИЕ

Программы обычно оперируют информацией, которая содержится в таблицах. В большинстве случаев эти таблицы представляют собой не просто бесформенную массу численных значений, а элементы данных с важными *структурными взаимосвязями.*

В простейшем виде таблица может выглядеть так, как простой линейный список элементов, а сведения о ее структурных свойствах можно получить, ответив на следующие вопросы. Какой элемент располагается в списке первым? Какой последним? Какие элементы предшествуют данному элементу, а какие следуют за ним? Сколько элементов содержится в списке? Ответив на такие вопросы, даже в этом простом случае можно получить богатую информацию о структуре данных (см. раздел 2.2).

В более сложных случаях таблица может иметь структуру двумерного массива (т. е. матрицы или решетки, состоящей из строк и столбцов) или даже n-мерного массива для n больше 2, древовидную структуру (tree structure) , которая представляет иерархические или разветвляющиеся связи, либо такую сложную многосвязную структуру (multilinked structure) с большим количеством соединений, которая устроена, как человеческий мозг.

Для правильного использования компьютера необходимо хорошо знать структурные взаимосвязи между данными, основные методы представления структур внутри компьютера, а также методы работы с ними.

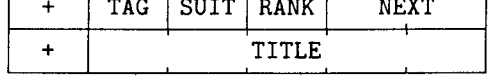
В данной главе приводятся наиболее важные сведения об информационных структурах: статические и динамические свойства структур разных типов, средства выделения памяти для хранения и представления структурированных данных, эффективные алгоритмы для создания, изменения, удаления информации о структуре данных, а также для доступа к ней. В ходе изучения материала на нескольких важных примерах будет проиллюстрировано применение этих методов для решения широкого круга задач. В примерах будут рассмотрены топологическая

сортировка, арифметика многочленов, моделирование дискретных систем, работа с разреженными матрицами, манипулирование алгебраическими формулами, а также создание компиляторов и операционных систем. Основное внимание здесь будет уделено представлению структуры *внутри* компьютера; ее преобразование между внешними и внутренними представлениями будет подробно рассмотрено в главах 9 и 10.

Большая часть представленного здесь материала часто называется обработкой списков, особенно с тех пор как было создано достаточно систем программирования, например LISP , предназначенных специально для работы со структурами общего типа под названием *Списки*. (Далее в этой главе слово "Список" с прописной начальной буквой будет использоваться для обозначения отдельного типа структуры, которая более подобно рассматривается в разделе 2.3.5.) Хотя в большинстве случаев системы обработки Списков очень полезны, они часто накладывают на работу программиста не всегда оправданные ограничения. В собственных программах обычно эффективнее непосредственно использовать предлагаемые в этой главе методы, подгоняя для конкретного приложения формат данных и алгоритмы их обработки. Многие разработчики программного обеспечения все еще считают, что методы обработки Списков очень сложны (и потому вынуждены использовать интерпретаторы сторонних разработчиков или уже готовые наборы процедур) и что обработка Списков может выполняться лишь некоторым строго определенным способом. Как будет показано ниже, ничего сверхъестественного, загадочного или трудного в работе со сложными структурами нет. Приведенные здесь методы являются частью арсенала каждого программиста, и их можно использовать при создании программ на языке ассемблера или на таких алгебраических языках, как FORTRAN, C и Java.

Методы работы с информационными структурами будут проиллюстрированы здесь на примере компьютера MIX. Читателю, которого не интересует подробная структура программ для компьютера MIX, следует изучить хотя бы способы, с помощью которых информация о структуре представляется в памяти компьютера MIX.

Здесь важно определиться с терминами и обозначениями, которые будут довольно часто использоваться. Информация в таблице состоит из набора *узлов* (*nodes*); некоторые авторы называют их записями (records) , объектами (entities) или цепочками (beads) . Иногда вместо термина "узел" будут использоваться термины "предмет" и "элемент". Каждый узел состоит из одного или нескольких последовательных слов в памяти компьютера, которые могут быть разделены на именованные части — *поля*. В простейшем случае узел представляет собой только одно слово и содержит только одно поле, охватывающее это слово целиком. В качестве другого и более интересного примера рассмотрим элементы таблицы, которая предназначена для представления игральных карт. Предположим, что узлы в ней состоят из двух слов, которые делятся на пять полей — TAG, SUIT, RANK, NEXT и TITLE:



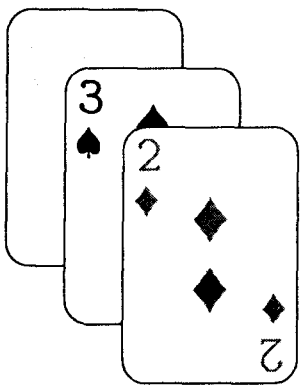
(1)

(Так выглядит формат содержимого двух слов в компьютере MIX. Следует напомнить, что слово в компьютере MIX состоит из пяти байтов и знака (см. раздел 1.3.1).)

В этом примере предполагается, что каждое слово имеет знак "+" (плюс). *Адрес* (*address*) узла, который также называется *связью* (*link*), *указателем* (*pointer*) или *ссылкой* (*reference*) на этот узел, является адресом его первого слова. Адрес часто указывается относительно некоторой базовой ячейки памяти, но в этой главе для простоты предполагается, что адрес является абсолютным.

Любое поле внутри узла может содержать числа, буквы, ссылки или нечто другое, заданное программистом. В приведенном выше примере колода карт для раскладывания пасьянса может быть представлена следующим образом: `TAG` = 1 означает, что карта обращена лицевой стороной вниз, `TAG` = 0 — лицевой стороной вверх; `SUIT` = 1, 2, 3 или 4 означает масть карты, т. е. трефы, бубны, черви или пики соответственно; `RANK` = 1, 2, ..., 13 означает ранг карты, т. е. туз, двойка, ..., король; `NEXT` является *связью* с картой, расположенной *под* данной картой в этой же колоде; `TITLE` представляет предназначенное для печати имя карты из пяти символов. Типичная колода карт может выглядеть так, как показано ниже.

Карты



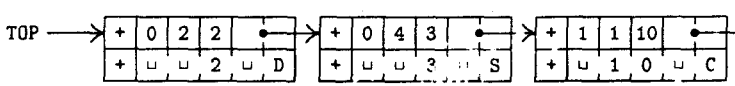
Представление в компьютере

100:	+	1	1	10	Λ
101:	+	␣	1	0	␣ C
386:	+	0	4	3	100
387:	+	␣	␣	3	␣ S
242:	+	0	2	2	386
243:	+	␣	␣	2	␣ D

(2)

Адреса карт в компьютерном представлении показаны здесь в виде чисел 100, 386 и 242, хотя в данном примере они могли бы быть любыми, поскольку каждая карта связана со следующей картой, расположенной под ней. Обратите внимание на особую связь "Λ" в узле 100. *Здесь прописная греческая буква "лямбда" обозначает пустую связь*, т. е. связь с несуществующим узлом. Пустая ссылка Λ используется в узле 100, так как десятка треф является самой нижней картой в колоде. В компьютере ссылка Λ представлена легко распознаваемым значением, которое не может быть адресом узла. Вообще, предположим, что по адресу 0 никаких узлов нет, тогда Λ практически всегда в программах для компьютера `MIX` может быть представлена как связь с нулевым значением адреса.

Введение понятия связи с другими элементами данных является чрезвычайно важной идеей в компьютерном программировании и ключевым способом представления сложных структур. При создании схемы представления узлов в компьютере связи обычно изображаются в виде стрелок. Тогда схема примера (2) будет выглядеть следующим образом:



(3)

При этом фактические адреса 242, 386 и 100 (которые в данном примере все равно не имеют большого значения) в схеме (3) не представлены. Используемое в электротехнике обозначение заземления здесь применяется для обозначения пустой связи в правой части схемы. Обратите внимание также, что в схеме (3) на самую верхнюю карту указывает стрелка TOP. Здесь TOP представляет собой *переменную связи (link variable)*, которая часто называется *указателем* (*pointer variable*), т. е. переменной, значением которой является связь. Все ссылки на узлы в программе определяются непосредственно с помощью переменных связи (или констант связи) либо косвенно с помощью полей связи в других узлах.

Теперь рассмотрим самую важную часть системы обозначений — обозначение ссылки на поле внутри некоторого узла. Оно выглядит достаточно просто, поскольку для этого нужно лишь привести имя данного поля и указать вслед за ним в скобках связь с нужным узлом. Например, для случаев (1)–(3) это можно сделать так, как показано ниже:

$$\begin{aligned} \texttt{RANK(100)} &= 10; \qquad \texttt{SUIT(TOP)} = 2; \\ \texttt{TITLE(TOP)} &= \text{“}\texttt{␣␣2␣D}\text{”}; \qquad \texttt{RANK(NEXT(TOP))} = 3. \end{aligned} \tag{4}$$

Читателю следует внимательно изучить эти примеры, поскольку такие обозначения полей будут применяться во многих других алгоритмах в настоящей и последующих главах. Чтобы пояснить смысл этой идеи, рассмотрим простой алгоритм, предназначенный для размещения с верхней стороны колоды новой карты с лицевой стороной, обращенной вверх. При этом допустим, что значение переменной связи NEWCARD содержит связь с новой картой.

A1. Установить NEXT(NEWCARD) $\leftarrow$ TOP. (Таким образом, значение поля связи в новой карте будет указывать на верхнюю карту колоды.)

A2. Установить TOP $\leftarrow$ NEWCARD. (TOP по-прежнему будет указывать на верхнюю карту колоды.)

A3. Установить TAG(TOP) $\leftarrow$ 0. (Карта обращена лицевой стороной вверх.) ▮

В другом примере рассмотрим алгоритм для вычисления текущего количества карт в колоде.

B1. Установить N $\leftarrow$ 0, X $\leftarrow$ TOP. (Здесь N является целочисленной переменной, а X — переменной связи.)

B2. Если X $= \Lambda$, прекратить выполнение алгоритма; при этом значение N будет равно количеству карт в колоде.

B3. Установить N $\leftarrow$ N + 1, X $\leftarrow$ NEXT(X) и вернуться к шагу B2. ▮

Обратите внимание на то, что в этих алгоритмах символьные имена используются для двух совершенно разных объектов: как имена *переменных* (TOP, NEWCARD, N, X) и как имена *полей* (TAG, NEXT). Их не следует путать. Если F — это имя поля и L $\neq \Lambda$ — связь, то F(L) — это переменная. Но сам по себе символ F не является переменной, поскольку не обладает никаким значением, если только оно не является непустой связью.

При обсуждении подробностей работы компьютера на низком уровне будут использоваться приведенные далее обозначения, которые применяются для преобразования хранимых значений и их адресов.

a) CONTENTS всегда обозначает поле длиной в одно слово в узле длиной в одно слово. Таким образом, CONTENTS(1000) — это значение, хранимое в памяти по адресу 1000, т. е. переменная с таким значением. Если V является переменной связи, то CONTENTS(V) — значение, на которое указывает V (а не само значение переменной связи V).

b) Если V является именем переменной, значение которой хранится в некоторой ячейке памяти, то LOC(V) обозначает адрес этой ячейки. Соответственно, если V является переменной, значение которой хранится в памяти в виде полного слова, то CONTENTS(LOC(V)) = V.

Эти обозначения можно легко преобразовать в код языка ассемблера MIXAL, хотя обозначения MIXAL выглядят несколько иначе. Значения переменных связи помещаются в индексные регистры, и, чтобы сослаться на нужное поле, необходимо использовать средства MIX для доступа к части поля. Например, приведенный выше алгоритм A можно записать следующим образом.

```
NEXT EQU  4:5          Определение полей NEXT
TAG  EQU  1:1              и TAG для ассемблера.
     LD1  NEWCARD      A1. rI1 ← NEWCARD.
     LDA  TOP              rA ← TOP.                (5)
     STA  0,1(NEXT)        NEXT(rI1) ← rA.
     ST1  TOP          A2. TOP ← rI1.
     STZ  0,1(TAG)     A3. TAG(rI1) ← 0.  ∎
```

Простота и эффективность выполнения этих операций на компьютере — вот основные причины использования концепции "связанной памяти".

Иногда приходится иметь дело с простой переменной, которая обозначает целый узел, а потому ее значение представляет не одно поле, а последовательность полей. В таком случае можно записать

$$\texttt{CARD} \leftarrow \texttt{NODE(TOP)}, \tag{6}$$

где NODE — такая же спецификация поля, как и CONTENTS, но относится она к целому узлу, а CARD является переменной, которая имеет структуру наподобие (1). Если узел имеет размер c слов, то обозначение (6) является сокращенной записью для c операций присвоения, выполняемых на низком уровне:

$$\texttt{CONTENTS(LOC(CARD)} + j\texttt{)} \leftarrow \texttt{CONTENTS(TOP} + j\texttt{)}, \quad 0 \le j < c. \tag{7}$$

Между языком ассемблера и обозначениями, используемыми в алгоритмах, есть важное отличие. Так как язык ассемблера близок к внутреннему языку компьютера, то используемые в программах MIXAL символы обозначают адреса, а не значения. Поэтому в левых столбцах кода (5) символ TOP на самом деле обозначает *адрес* в памяти, по которому находится указатель на самую верхнюю карту в колоде, а в выражениях (6) и (7) и комментариях в (5) справа он является *значением* TOP, а именно — адресом узла самой верхней карты. Такое различие между языком ассемблера и языком высокого уровня часто является источником ошибок в работе начинающих программистов, поэтому читателю настоятельно рекомендуется выполнить упр. 7, а также другие упражнения из данного раздела для закрепления навыков работы с введенными здесь обозначениями.

УПРАЖНЕНИЯ

1. [*04*] В ситуации, показанной на схеме (3), каким будет значение выражения (a) SUIT(NEXT(TOP)); (b) NEXT(NEXT(NEXT(TOP)))?

2. [*10*] В приведенном выше разделе сказано, что во многих случаях CONTENTS(LOC(V)) = V. При каких условиях LOC(CONTENTS(V)) = V?

3. [*11*] Предложите алгоритм, обратный алгоритму A: он должен удалить самую верхнюю карту колоды (если колода не пуста) и задать ссылку NEWCARD для адреса этой карты.

4. [*18*] Предложите алгоритм, аналогичный алгоритму A, но новая карта должна располагаться *лицевой стороной вниз* в *нижней части* колоды. (Колода может быть пустой.)

▶ **5.** [*21*] Предложите алгоритм, обратный алгоритму из упр. 4. Допустим, что колода не пуста и самая нижняя карта в ней расположена лицевой стороной вниз. Предложенный вами алгоритм должен удалить эту карту и указать ее адрес в переменной связи NEWCARD. (Данный алгоритм при раскладывании пасьянсов иногда называется мошенничеством.)

6. [*06*] В примере с игральными картами предположим, что CARD — это имя переменной, значением которой является весь узел, как показано в (6). Операция CARD ← NODE(TOP) устанавливает поля CARD равными соответствующим полям самой верхней карты колоды. Какое из перечисленных ниже обозначений будет соответствовать масти самой верхней карты после выполнения этой операции:

(a) SUIT(CARD); (b) SUIT(LOC(CARD)); (c) SUIT(CONTENTS(CARD)); (d) SUIT(TOP)?

▶ **7.** [*04*] В приведенном выше примере программы (5) для компьютера MIX переменная связи с верхней картой хранится в слове компьютера MIX, которое на языке ассемблера называется TOP. Какой из приведенных вариантов кода для заданной структуры поля (1) позволит занести значение NEXT(TOP) в регистр A? Объясните, почему другой вариант неверен.

a)
```
LDA  TOP(NEXT)
```

b)
```
LD1  TOP
LDA  0,1(NEXT)
```

▶ **8.** [*18*] Напишите программу для компьютера MIX, соответствующую алгоритму B.

9. [*23*] Напишите программу для компьютера MIX, которая печатает названия карт из данной колоды, начиная с самой верхней карты и размещая их по одной в каждой строке со скобками вокруг карт, повернутых лицевой стороной вниз.

2.2. ЛИНЕЙНЫЕ СПИСКИ

2.2.1. Стеки, очереди и деки

СТРУКТУРА ДАННЫХ обычно гораздо богаче структуры, действительно необходимой для их представления в компьютере. Например, каждый узел игральной карты из предыдущего раздела имеет поле NEXT для указания карты, расположенной в колоде под ней. Однако до сих пор не было указано ни одного способа определения карты, которая расположена *над* данной картой (если таковая вообще имеется), или определения колоды, в которой находится данная карта. И, конечно же, совсем не были учтены характерные черты *реальных* игральных карт: детали оформления рубашки, расположение по отношению к другим объектам в данной комнате, молекулярное строение карт и т. д. Вполне вероятно, что такая информация очень важна для некоторых компьютерных приложений, но совершенно очевидно, что неразумно сохранять абсолютно *всю* информацию о структуре в каждой ситуации. Действительно, в большинстве случаев для использования игральных карт достаточно лишь некоторых сведений из приведенных в первом примере. Например, поле TAG, в котором содержится информация о том, как ориентирована лицевая поверхность карты (вверх или вниз), часто является избыточным.

В каждом конкретном случае следует принять отдельное решение о том, насколько полно информация о структуре данных должна быть представлена в таблицах и как получить доступ к каждому элементу информации. Для принятия подобных решений необходимо знать операции, которые нужно выполнить над этими данными. Поэтому для решения каждой задачи в настоящей главе будут рассмотрены структура данных и набор операций, которые выполняются над данными. Способ представления данных в компьютере зависит не только от способа их функционирования, но и от присущих им свойств. Действительно, при решении общих задач проектирования в основном учитываются способ функционирования и форма данных.

Для иллюстрации этой идеи рассмотрим задачу проектирования аппаратного обеспечения. Память компьютера подразделяется на память с произвольным доступом (random access memory), подобную оперативной памяти компьютера MIX; память с доступом только для чтения (read-only memory), которая предназначается для хранения неизменяемых данных; вторичную память большого объема (secondary bulk memory), подобную дисковым накопителям компьютера MIX, которые позволяют хранить огромный объем информации но с малой скоростью доступа к ней; ассоциативную память (associative memory или, точнее, content-addressed memory), в которой доступ к информации осуществляется по значению, а не по адресу, и т. д. Назначение каждого типа памяти столь велико, что оно указано в ее названии. Хотя все эти устройства являются компонентами "памяти", их функции в значительной степени влияют на их структуру и стоимость.

Линейный список представляет собой последовательность $n \geq 0$ узлов $\mathtt{X}[1]$, $\mathtt{X}[2]$, ..., $\mathtt{X}[n]$, важнейшей структурной особенностью которой является такое расположение элементов списка один относительно другого, как будто они находятся на одной линии. Иначе говоря, в такой структуре должно соблюдаться следующее условие: если $n > 0$ и $\mathtt{X}[1]$ является первым узлом, а $\mathtt{X}[n]$ — последним, то k-й узел $\mathtt{X}[k]$ следует за $\mathtt{X}[k-1]$ и предшествует узлу $\mathtt{X}[k+1]$ для всех $1 < k < n$.

С линейными списками могут выполняться следующие операции.

i) Получение доступа к k-му узлу списка для проверки и/или изменения содержимого его полей.
ii) Вставка нового узла сразу после или до k-го узла.
iii) Удаление k-го узла.
iv) Объединение в одном списке двух (или более) линейных списков.
v) Разбиение линейного списка на два (или более) списка.
vi) Создание копии линейного списка.
vii) Определение количества узлов в списке.
viii) Сортировка узлов в порядке возрастания значений в определенных полях этих узлов.
ix) Поиск узла с заданным значением в некотором поле.

В операциях (i)–(iii) очень важны особые случаи, когда $k = 1$ и $k = n$, поскольку доступ к первому и последнему элементам линейного списка можно организовать гораздо проще, чем к любому другому элементу списка. Операции (viii) и (ix) в данной главе рассматриваться не будут, так как они обсуждаются в главах 5 и 6 соответственно.

В одном компьютерном приложении редко используются сразу все девять типов операций в общей их формулировке. Поэтому линейные списки могут иметь самые разные представления в зависимости от класса операций, которые наиболее часто должны с ними выполняться. Достаточно трудно создать единое представление линейных списков, при котором эффективно выполнялись бы все эти операции. Например, сравнительно сложно организовать доступ к k-му узлу длинного списка для произвольно выбранного k, если одновременно необходимо выполнять вставку и удаление элементов в середине списка. Поэтому нужно различать разные типы линейных списков в зависимости от выполняемых с ними основных операций так же, как следует различать типы памяти компьютера, которые определяются ее конкретным назначением.

Линейные списки, в которых операции вставки, удаления и доступа к значениям чаще всего выполняются в первом или последнем узле, получили следующие специальные названия.

Стек — это линейный список, в котором все операции вставки и удаления (и, как правило, операции доступа к данным) выполняются только на одном из концов списка.

Очередь или *односторонняя очередь* — это линейный список, в котором все операции вставки выполняются на одном из концов списка, а все операции удаления (и, как правило, операции доступа к данным) — на другом.

Дек или *двусторонняя очередь* (double-ended queue) это линейный список, в котором все операции вставки и удаления (и, как правило, операции доступа к данным) выполняются на обоих концах списка.

Дек, следовательно, является более общим вариантом стека или очереди. Он обладает некоторыми общими свойствами с рассмотренной выше колодой карт, к тому же на английском языке эти термины созвучны. Кроме того, следует различать деки *с ограниченным выводом* (*output-restricted deque*) и *с ограниченным вводом* (*input*

restricted deque), в которых операции удаления и вставки элементов соответственно выполняются только на одном из концов.

В некоторых дисциплинах слово "очередь" используется в более широком смысле, например, при описании любого типа списка, для которого выполняются операции вставки и удаления. Тогда упомянутые выше особые случаи следует характеризовать как разные "правила упорядочения" (или "очередности обслуживания"). В этой книге термин "очередь" предполагается использовать только в узком смысле, по аналогии с обычной упорядоченной очередью людей, ожидающих обслуживания.

Механизм работы стека можно представить проще, если воспользоваться аналогией с железнодорожным разъездом, которая предложена Э. В. Дейкстрой (рис. 1). Соответствующая схема для дека показана на рис. 2.

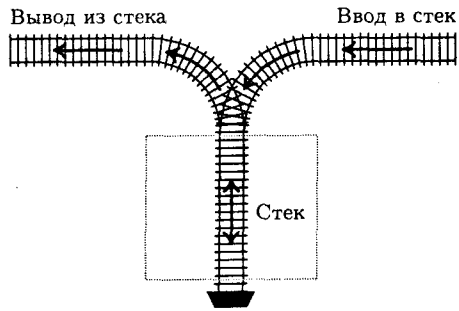


Рис. 1. Схема стека в виде железнодорожного разъезда.

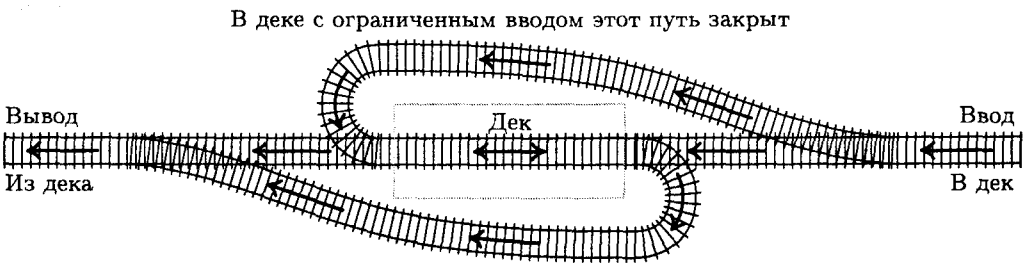


Рис. 2. Схема дека в виде железнодорожного разъезда.

При выполнении операции удаления в стеке прежде всех удаляется "младший" объект списка, т. е. объект, который был вставлен в список позже других. В очереди удаление выполняется наоборот: раньше всех удаляется "старший" объект, поскольку узлы покидают очередь в том же порядке, в котором они вставлялись.

Многие исследователи независимо пришли к выводу о важности стеков и очередей, а потому присвоили им иные собственные имена. Так, стеки часто называют магазинными списками (push-down lists), реверсивными хранилищами (reversion storages), магазинами (cellars), вложенными хранилищами (nesting stores), кучами (piles), дисциплинами обслуживания в обратном порядке (last-in-first-out lists — LIFO lists) и даже флюгерными списками (yo-yo lists). Очереди часто называют

циклическими хранилищами (circular stores) или дисциплинами обслуживания в порядке поступления (first-in-first-out lists — FIFO lists). Термины "LIFO" и "FIFO" многие годы употреблялись бухгалтерами для обозначения метода оценки и продажи товаров. Еще один термин, "стеллаж" (shelf), применялся для деков с ограниченным выводом. Аналогично деки с ограниченным вводом назывались также свитками (scrolls или rolls). Такое разнообразие имен само по себе интересно уже тем, что оно свидетельствует о важности этих концепций. Постепенно слова "стек" и "очередь" вошли в стандартную терминологию , и только термин "магазинный список" все еще остается сравнительно популярным, особенно в теории автоматов.

На практике довольно часто приходится иметь дело со стеками. Например, анализируя набор данных, часто требуется составить список возникающих исключительных ситуаций или действий, которые необходимо выполнить впоследствии. После анализа исходного набора данных можно перейти к выполнению этих действий с помощью списка, удаляя его элементы до полного опустошения списка. (Примером такой ситуации является задача о "точке перевала", которая описана в упр. 1.3.2–10.) Для организации подобного списка удобно использовать стек или очередь (хотя стек гораздо удобнее). При решении задач мы практически всегда мысленно организуем некое подобие "стека", в котором одна задача порождает другую, а другая, в свою очередь, — следующую. Так задачи накапливаются и соответственно разрешаются одна за другой. Аналогично процесс входа в подпрограммы и выхода из них во время выполнения компьютерной программы осуществляется подобно стеку. Стеки особенно полезны при обработке языков с вложенной структурой, например для языков программирования, арифметических выражений и конструкций "Schachtelsätze" немецкого языка. Вообще, стеки чаще всего встречаются при работе с явными и неявными рекурсивными алгоритмами, которые более подробно рассматриваются в главе 8.

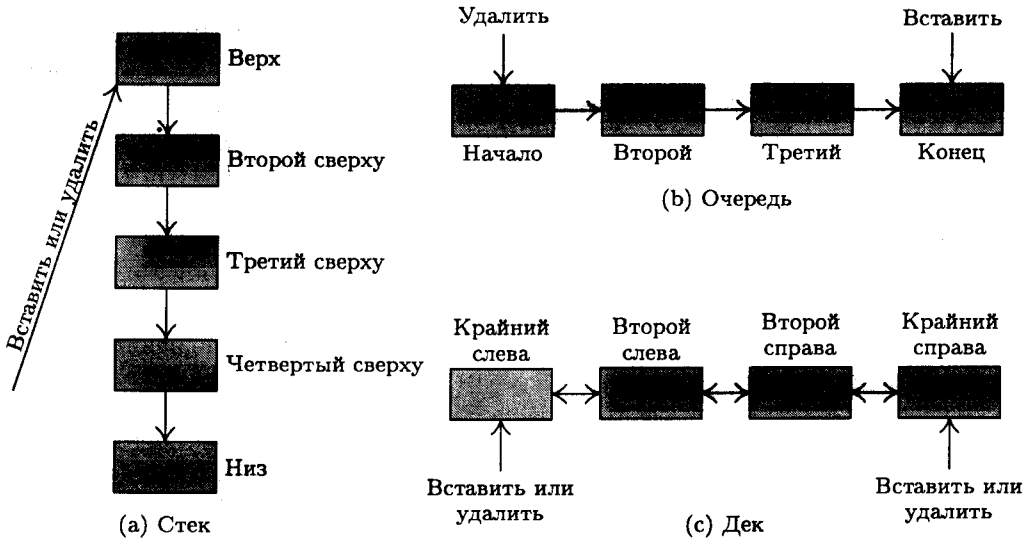


Рис. 3. Три наиболее важных класса линейных списков.

Для описания алгоритмов обработки этих структур обычно используется специальная терминология. Например, объект кладут на *верх* стека или снимают верхний элемент стека (рис. 3, (a)). Доступ к элементу, находящемуся *внизу* стека, наиболее затруднителен, и его нельзя удалить до тех пор, пока не будут удалены все остальные элементы стека. (Часто говорят, что объект *проталкивается* (*push*) *вниз* стека и *выталкивается* (*pop*) на *верх* стека при удалении самого верхнего элемента. Эта терминология возникла от аналогии со стопкой тарелок в кафе. Краткость английского написания слов "протолкнуть" (push) и "вытолкнуть" (pop) обладает определенными преимуществами, но эти термины часто неверно трактуются, как движение всего списка целиком в памяти компьютера. На самом деле физически ничто никуда не проталкивается, а объекты всего лишь добавляются сверху стека точно так, как при укладке сена в стоге или коробок в стопке.) По отношению к очередям применяют понятия *начало* (*front*) и *конец* (*rear*) очереди. Объекты вставляются в конце очереди и проталкиваются по ней до тех пор, пока не достигнут начала очереди (рис. 3, (b)). При работе с деками используют понятия *левый* (*left*) и *правый* (*right*) концы (рис. 3, (c)). Концепции верха, низа, начала и конца иногда применяют по отношению к декам, которые используются в качестве стеков или очередей, но нет никаких стандартных соглашений в отношении того конца, с которого они должны располагаться: с правого или левого.

Таким образом, мы выяснили, что при создании алгоритмов достаточно удобно применять все богатое разнообразие языка: "верх-низ" (up-down) — для стеков, "ожидание в очереди" (waiting in line) — для очередей и "слева-справа" (left-right) — для деков.

При работе со стеками и очередями удобно использовать некоторые дополнительные обозначения. Например, обозначение

$$A \Leftarrow x \tag{1}$$

указывает, что элемент x *вставлен* сверху стека A (если A — это стек) или элемент x *вставлен* в конце очереди (если A — это очередь). Аналогично

$$x \Leftarrow A \tag{2}$$

используется для обозначения того, что переменная x приравнивается к значению верхнего элемента стека A или начального элемента очереди A, т. е. это значение, которое *удаляется* из A. Обозначение (2) не имеет смысла, если A пусто, т. е. когда A не содержит значений.

Если A — непустой стек, то

$$\mathrm{top}(A) \tag{3}$$

можно использовать для обозначения самого верхнего его элемента.

УПРАЖНЕНИЯ

1. [*06*] Дек с ограниченным вводом является линейным списком, в котором элементы могут вставляться на одном конце, а удаляться — с любого конца. Очевидно, что дек может функционировать, как стек или очередь, если его элементы всегда будут удаляться с одного из двух его концов. Может ли дек с ограниченным выводом функционировать, как стек или очередь?

▸ **2.** [*15*] Допустим, что четыре вагона с номерами 1, 2, 3 и 4 (слева направо) расположены со стороны ввода вагонов железнодорожных путей (см. рис. 1). Предположим, что выполняется такая последовательность действий (которая совпадает с направлением стрелок на этой схеме и не допускает "перепрыгивания" через вагоны): (i) поместить вагон 1 в стек; (ii) поместить вагон 2 в стек; (iii) вывести вагон 2 из стека; (iv) поместить вагон 3 в стек; (v) поместить вагон 4 в стек; (vi) вывести вагон 4 из стека; (vii) вывести вагон 3 из стека; (viii) вывести вагон 1 из стека.

После выполнения этих действий исходный порядок вагонов, 1234, станет иным, 2431. *Целью этого и следующих упражнений является выяснение, какие перестановки допустимы в стеках, очередях и деках.*

Можно ли выполнить такую перестановку шести пронумерованных вагонов, чтобы из исходного порядка 123456 получить порядок 325641? Можно ли поменять порядок вагонов так, чтобы он был равен 154623? (Если можно, то как именно?)

3. [*25*] Действия (i)–(viii) в предыдущем упражнении можно записать в более кратком виде с помощью кода SSXSSXXX, где S обозначает "ввести вагон в стек", а X — "вывести вагон из стека". Некоторые последовательности действий S и X бессмысленны, поскольку на соответствующих путях может совсем не быть вагонов. Например, последовательность действий SXXSSXXS не может быть выполнена, так как предполагается, что в исходном состоянии стек пуст.

Назовем последовательность действий S и X *допустимой*, если она содержит n действий S и n действий X и если в ней нет никаких бессмысленных действий. Сформулируйте правило, с помощью которого было бы легко различать допустимые и недопустимые последовательности действий. Покажите, что различные допустимые последовательности действий приводят к разным перестановкам.

4. [*M34*] Найдите простую формулу для a_n, т. е. для количества перестановок среди n элементов, которые могут быть получены с помощью стека из упр. 2.

▸ **5.** [*M28*] Покажите, что с помощью стека можно получить перестановку $p_1 p_2 \dots p_n$ исходной последовательности $12 \dots n$ только в случае, если не существует таких индексов $i < j < k$, что $p_j < p_k < p_i$.

6. [*00*] Рассмотрим задачу из упр. 2, в которой очередь используется вместо стека. Какие перестановки $12 \dots n$ можно получить с помощью очереди?

▸ **7.** [*25*] Рассмотрим задачу из упр. 2, в которой вместо стека используется дек. (a) Найдите такую перестановку последовательности 1234, которая может быть получена с помощью дека с ограниченным вводом, но не может быть получена с помощью дека с ограниченным выводом. (b) Найдите такую перестановку последовательности 1234, которая может быть получена с помощью дека с ограниченным выводом, но не может быть получена с помощью дека с ограниченным вводом. [Как следствие из (a) и (b) между деками с ограниченным вводом и выводом существует определенная разница.] (c) Найдите такую перестановку последовательности 1234, которая не может быть получена ни с помощью дека с ограниченным вводом, ни с помощью дека с ограниченным выводом.

8. [*22*] Существуют ли какие-либо перестановки $12 \dots n$, которые не могут быть получены с помощью дека, не имеющего ни ограниченного ввода, ни ограниченного вывода?

9. [*M20*] Пусть b_n — количество перестановок среди n элементов, которые могут быть получены с помощью дека с ограниченным вводом. (Обратите внимание, что $b_4 = 22$, как показано в упр. 7.) Покажите, что b_n также является количеством перестановок среди n элементов, которые могут быть получены с помощью дека с ограниченным *выводом*.

10. [*M25*] (См. упр. 3.) Пусть S, Q и X обозначают соответственно операции ввода элемента слева, ввода элемента справа и вывода элемента слева в деке с ограниченным выводом.

Например, применив последовательность действий QQXSXSXX к исходной последовательности символов 1234, получим 1342. Последовательность действий SXQSXSXX приведет к такому же результату.

Найдите такое определение понятия *допустимой* последовательности символов S, Q и X, чтобы выполнялось следующее требование: каждая перестановка n элементов, которую можно получить с помощью дека с ограниченным выводом, должна соответствовать только одной допустимой последовательности.

▸ **11.** [*M40*] Из упр. 9 и 10 получаем, что b_n является количеством допустимых последовательностей длины $2n$. Найдите производящую функцию $\sum_{n\geq 0} b_n z^n$ в "замкнутом виде".

12. [*HM34*] Вычислите асимптотические значения величин a_n и b_n из упр. 4 и 11.

13. [*M48*] Сколько перестановок среди n элементов можно получить с помощью дека? [В статье Розенстиля и Таржана, *J. Algorithms* **5** (1984), 389–390, приводится алгоритм, в котором за $O(n)$ шагов выясняется допустимость данной перестановки.]

▸ **14.** [*26*] Предположим, что в качестве структур данных используются только стеки. Как наиболее эффективным образом воплотить очередь с помощью двух стеков?

2.2.2. Последовательное распределение

Наиболее простой и естественный способ хранения линейного списка в памяти компьютера заключается в расположении элементов в последовательных ячейках, при котором один узел списка следует сразу же за другим. Тогда

$$\texttt{LOC(X[}j+1\texttt{])} = \texttt{LOC(X[}j\texttt{])} + c,$$

где c — количество слов в одном узле. (Обычно $c = 1$. Если $c > 1$, то в некоторых случаях удобнее разбить единый список на c "параллельных" списков таким образом, чтобы k-е слово узла `X[`j`]` хранилось на фиксированном расстоянии от первого слова `X[`j`]`, которое зависит от k. Однако в дальнейшем предположим, что смежные группы по c слов образуют единый узел.) Вообще,

$$\texttt{LOC(X[}j\texttt{])} = \texttt{L}_0 + cj, \tag{1}$$

где $\texttt{L}_0$ — константа, которая называется *базовым адресом* (*base address*), т. е. является адресом некоего узла `X[0]`.

Такой способ представления линейного списка настолько очевиден и хорошо известен, что его более глубокое изучение может показаться вовсе ненужным. Однако, как будет показано в данной главе, существует несколько других, более "изощренных" методов представления списков. Поэтому, прежде чем рассматривать более сложные случаи, было бы разумно изучить возможности этого простого способа. Очень важно ясно представлять себе как его недостатки, так и преимущества.

Последовательное распределение очень удобно при работе со *стеком*. Для этого достаточно иметь переменную `T`, которая называется *указателем стека* (*stack pointer*). Если стек пуст, то `T` = 0. Для ввода нового элемента `Y` в стек следует выполнить такие действия:

$$\texttt{T} \leftarrow \texttt{T} + 1; \qquad \texttt{X[T]} \leftarrow \texttt{Y}. \tag{2}$$

А если стек не пуст, можем положить `Y` равным верхнему узлу и удалить этот узел, выполнив действия, обратные действиям (2):

$$\texttt{Y} \leftarrow \texttt{X[T]}; \qquad \texttt{T} \leftarrow \texttt{T} - 1. \tag{3}$$

(Внутри компьютера эффективнее было бы оперировать значениями cT вместо T в силу соотношений (1). Такие изменения можно выполнить достаточно легко, поэтому дальнейшие рассуждения продолжим, полагая, что $c = 1$.)

Представление *очереди* или *дека* организовано несколько сложнее. Очевидным решением могло бы быть применение двух указателей, F и R (для начала и конца очереди), где $F = R = 0$, если очередь пуста. Тогда вставка элемента с конца очереди может быть выполнена с помощью следующих действий:

$$R \leftarrow R + 1; \qquad X[R] \leftarrow Y. \tag{4}$$

Удаление же узла в начале очереди (F указывает на ячейку, расположенную перед начальным элементом очереди) может быть выполнено с помощью других действий:

$$F \leftarrow F + 1; \qquad Y \leftarrow X[F]; \qquad \text{если } F = R, \text{ то } F \leftarrow R \leftarrow 0. \tag{5}$$

Следует отметить, что, если при такой организации работы очереди R всегда опережает F (т. е. в очереди всегда имеется хотя бы один узел), элементы таблицы X[1], X[2], . . , X[1000], ... будут последовательно заниматься вплоть до бесконечности, что приведет к расточительному использованию памяти. Следовательно, простой метод на основе операций (4) и (5) нужно использовать только тогда, когда известно, что F настигает R достаточно регулярно, например если все удаления выполняются скачкообразно с полным опустошением очереди.

Чтобы разрешить задачу переполнения памяти при такой организации очереди, можно выделить M узлов X[1],...,X[M], неявно образующих замкнутое кольцо, в котором за X[M] следует X[1]. Тогда описанные выше действия (4) и (5) будут выглядеть так:

$$\text{если } R = M, \text{ то } R \leftarrow 1, \text{ в противном случае } R \leftarrow R + 1; \qquad X[R] \leftarrow Y; \tag{6}$$

$$\text{если } F = M, \text{ то } F \leftarrow 1, \text{ в противном случае } F \leftarrow F + 1; \qquad Y \leftarrow X[F]. \tag{7}$$

Фактически циклическая организация очереди уже встречалась ранее, при описании буферов ввода-вывода в разделе 1.4.4.

До сих пор эти рассуждения были далеки от реальности, поскольку неявно предполагалось отсутствие каких-либо нежелательных ситуаций. Например, при удалении узла из стека или очереди допускалось, что в них остается по крайней мере еще один узел. А при вставке узла в стек или очередь допускалось, что для него найдется свободное место в памяти. Но, очевидно, при использовании метода на основе действий (6) и (7) предполагается, что в очереди может быть не более M узлов, а при использовании (2)–(5) значения T и R в данной программе не могут превышать определенный максимум. В общем случае, когда такие ограничения не выполняются автоматически, следует предпринять следующие действия:

$$X \Leftarrow Y \text{ (вставка в стек):} \begin{cases} T \leftarrow T + 1; \\ \text{если } T > M, \text{ то OVERFLOW}; \\ X[T] \leftarrow Y. \end{cases} \tag{2, а}$$

$$Y \Leftarrow X \text{ (удаление из стека):} \begin{cases} \text{если } T = 0, \text{ то UNDERFLOW}; \\ Y \leftarrow X[T]; \\ T \leftarrow T - 1. \end{cases} \tag{3, а}$$

$$\texttt{X} \Leftarrow \texttt{Y}\ \text{(вставка в очередь):}\ \begin{cases} \text{если } \texttt{R} = \texttt{M}, \text{ то } \texttt{R} \leftarrow 1, \\ \qquad \text{в противном случае } \texttt{R} \leftarrow \texttt{R} + 1; \\ \text{если } \texttt{R} = \texttt{F}, \text{ то } \texttt{OVERFLOW}; \\ \texttt{X[R]} \leftarrow \texttt{Y}. \end{cases} \qquad (6,\ \text{а})$$

$$\texttt{Y} \Leftarrow \texttt{X}\ \text{(удаление из очереди):}\ \begin{cases} \text{если } \texttt{F} = \texttt{R}, \text{ то } \texttt{UNDERFLOW}; \\ \text{если } \texttt{F} = \texttt{M}, \text{ то } \texttt{F} \leftarrow 1, \\ \qquad \text{в противном случае } \texttt{F} \leftarrow \texttt{F} + 1; \\ \texttt{Y} \leftarrow \texttt{X[F]}. \end{cases} \qquad (7,\ \text{а})$$

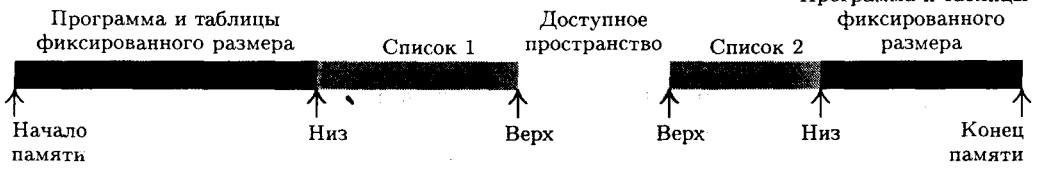
Здесь предполагается, что набор узлов `X[1]`, ..., `X[M]` образует общее пространство, доступное для организации списка. А обозначения `OVERFLOW` (переполнение) и `UNDERFLOW` (отсутствие) означают избыток или недостаток элементов. Теперь при использовании условий (6, а) и (7, а) начальное условие $\texttt{F} = \texttt{R} = 0$ для указателей очереди будет недействительным, поскольку избыток элементов (`OVERFLOW`) не будет зафиксирован при $\texttt{F} = 0$. Поэтому в качестве начального условия следует, например, выбрать $\texttt{F} = \texttt{R} = 1$.

Читателю настоятельно рекомендуется выполнить упр. 1, в котором обсуждается нетривиальный аспект этого простого механизма организации очереди.

В таком случае возникает вопрос: "Что нужно сделать при недостатке (`UNDERFLOW`) или избытке (`OVERFLOW`) элементов? Недостаток (`UNDERFLOW`) возникает при попытке удалить несуществующий элемент. Обычно эта ситуация рассматривается вовсе не как ошибка, а как некое условие с определенным смыслом, которое может быть использовано для управления потоком выполнения программы. Например, можно удалять элементы до тех пор, пока не наступит их недостаток (`UNDERFLOW`). Однако избыток (`OVERFLOW`) обычно свидетельствует об ошибке. Это значит, что таблица уже полностью заполнена, хотя в нее еще следует ввести некоторую информацию. При возникновении такой ситуации сначала обычно поступает сообщение о том, что программа не может продолжать работу, поскольку исчерпана емкость хранилища, а затем выполнение программы прекращается.

Конечно, вряд ли стоит прерывать выполнение программы в случае возникновения избытка (`OVERFLOW`), когда переполняется только один список, а в других списках той же программы еще остается достаточно свободного места. Выше предполагалось, что в программе имеется всего один список. Тем не менее довольно часто на практике используются программы с несколькими стеками, причем размер каждого из них может динамически изменяться. В таком случае не следует накладывать какое-либо ограничение на максимальный размер каждого стека, поскольку его размер непредсказуем. Даже если для каждого стека будет определен его максимальный размер, вряд ли возникнет ситуация, когда одновременно переполнятся *все* стеки.

Сосуществование всего двух списков с изменяемой длиной можно довольно просто организовать за счет роста их навстречу друг другу, как показано ниже.



Здесь список 1 расширяется вправо, а список 2 (хранящийся в обратном порядке) — влево. Избыток (OVERFLOW) не будет происходить до тех пор, пока общий размер обоих списков не будет превышать размер всего доступного пространства в памяти компьютера. Такие списки могут независимо расширяться и сжиматься так, что фактический максимальный размер каждого из них может существенно превышать половину всего доступного им пространства. Такая схема использования памяти часто применяется на практике.

Однако довольно просто можно убедиться в том, что нельзя организовать хранение в памяти компьютера трех и более последовательных списков с изменяемой длиной и соблюсти следующие условия: (a) избыток (OVERFLOW) должен происходить только тогда, когда общий размер всех списков превышает размер доступного для них пространства, (b) "нижний" элемент каждого списка должен иметь фиксированное место расположения. Проблема эффективного распределения пространства памяти для хранения списков становится очень важной при работе с десятью или более списками с изменяемыми размерами. Причем такие ситуации являются довольно обычными. В подобном случае для удовлетворения условия (a) придется пренебречь условием (b), т. е. позволить "нижним" элементам изменять их расположение. Это значит, что адрес L_0 в уравнении (1) уже не является постоянным, причем теперь нельзя будет делать ссылки на таблицу с помощью абсолютного адреса, поскольку все ссылки должны указываться относительно базового адреса L_0. В компьютере MIX код вставки I-го однословного узла в регистр A, который выглядел как

```
LD1  I
LDA  L0,1
```

, теперь будет, например, таким:

```
LD1  I
LDA  BASE(0:2)
STA  *+1(0:2)
LDA  *,1
```

, (8)

где BASE содержит $\boxed{L_0 \mid 0 \mid 0 \mid 0}$. Подобная относительная адресация, очевидно, выполняется дольше, чем адресация по фиксированному базовому адресу, но не намного, если в компьютере MIX реализована возможность "косвенной адресации" (indirect addressing) (см. упр. 3).

Особенно важным является случай, когда каждый список с изменяемой длиной является стеком. Тогда код может быть таким же эффективным, как и прежде, поскольку в любой момент приходится иметь дело только с верхним элементом каждого стека. Алгоритм вставки и удаления элементов в n стеках будет выглядеть следующим образом (здесь BASE$[i]$ и TOP$[i]$ — переменные связи i-го стека, а каждый узел составляет одно слово).

Вставка: TOP$[i] \leftarrow$ TOP$[i] + 1$; если TOP$[i] >$ BASE$[i+1]$, то OVERFLOW; в противном случае CONTENTS(TOP$[i]) \leftarrow$ Y. (9)

Удаление: если TOP$[i] =$ BASE$[i]$, то UNDERFLOW; в противном случае Y $\leftarrow$ CONTENTS(TOP$[i]$), TOP$[i] \leftarrow$ TOP$[i] - 1$. (10)

BASE$[i+1]$ является базовым адресом $(i+1)$-го стека. Условие TOP$[i] =$ BASE$[i]$ означает, что стек i пуст.

В алгоритме (9) условие избытка OVERFLOW уже не является таким критичным, как прежде. Теперь можно "переупаковать память", предоставляя больше свободного пространства для переполняющейся таблицы за счет незанятого пространства

из незаполненных таблиц. Можно использовать сразу несколько способов переупаковки, причем некоторые из них будут рассмотрены подробно, так как они очень важны для организации последовательного упорядочения линейных списков. Сначала рассмотрим наиболее простые способы, а затем — некоторые альтернативные им варианты.

Предположим, что существует n стеков и что значения BASE[i] и TOP[i] обрабатываются согласно алгоритмам (9) и (10). Предполагается также, что эти стеки совместно используют общую область памяти, состоящую из L ячеек, удовлетворяющих условию $L_0 < L \le L_\infty$. (Здесь L_0 и L_∞ — константы, которые задают общий размер доступной памяти в словах.) Начнем с рассмотрения случая, когда все стеки пусты, т. е.

$$\texttt{BASE}[j] = \texttt{TOP}[j] = L_0 \qquad \text{для } 1 \le j \le n. \tag{11}$$

Предположим также, что BASE[$n+1$] $= L_\infty$, и тогда алгоритм (9) будет применим для $i = n$.

При переполнении (OVERFLOW) стека i возможны три варианта развития событий.

a) Найдем наименьшее k, для которого $i < k \le n$ и TOP[k] $<$ BASE[$k+1$], если такое k существует. Теперь переместим все элементы на один узел *вверх*:

$$\texttt{CONTENTS}(L+1) \leftarrow \texttt{CONTENTS}(L) \text{ для } \texttt{TOP}[k] \ge L > \texttt{BASE}[i+1].$$

(Во избежание утраты информации это следует делать в порядке убывания, а не возрастания значений L. Если TOP[k] $=$ BASE[$i+1$], такие перемещения выполнять не потребуется.) Наконец, BASE[j] $\leftarrow$ BASE[j] $+ 1$ и TOP[j] $\leftarrow$ TOP[j] $+ 1$ для $i < j \le k$.

b) Допустим, что ни одно k не удовлетворяет условию (a), но существует такое *наибольшее* k, для которого $1 \le k < i$ и TOP[k] $<$ BASE[$k+1$]. Переместим все элементы на один узел *вниз*:

$$\texttt{CONTENTS}(L-1) \leftarrow \texttt{CONTENTS}(L) \text{ для } \texttt{BASE}[k+1] < L \le \texttt{TOP}[i].$$

(Это нужно сделать только в порядке возрастания значений L.) Затем BASE[j] $\leftarrow$ BASE[j] $- 1$ и TOP[j] $\leftarrow$ TOP[j] $- 1$ для $k < j \le i$.

c) Допустим, что TOP[k] $=$ BASE[$k+1$] для всех $k \ne i$. Тогда очевидно, что для нового элемента стека нельзя найти свободное пространство, и поэтому работу программы придется прервать.

На рис. 4 показан пример конфигурации памяти для случая, когда $n = 4$, $L_0 = 0$, $L_\infty = 20$, по окончании последовательного выполнения действий

$$I_1^* \; I_1^* \; I_4 \; I_2^* \; D_1 \; I_3^* \; I_1 \; I_1^* \; I_2^* \; I_4 \; D_2 \; D_1. \tag{12}$$

(Здесь I_j и D_j относятся к операциям вставки и удаления в стеке j, а звездочка обозначает переполнение (OVERFLOW). При этом предполагается, что в начальный момент для стеков 1–3 пространство в памяти не выделялось.)

Ясно что в начальный момент многих ситуаций переполнения стеков можно избежать, если выбрать оптимальные начальные условия вместо выделения сразу всего доступного пространства в памяти для n-го стека, как предлагается в условии (11). Например, если предполагается, что размеры всех стеков равны, то следует

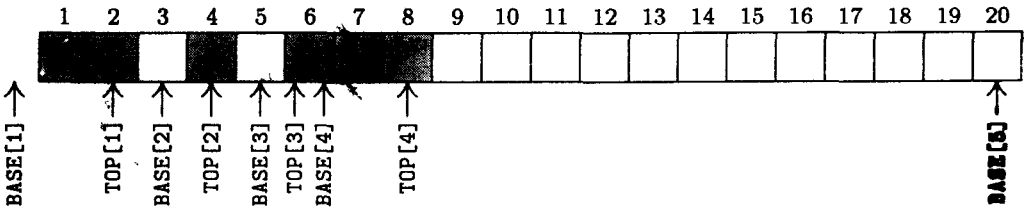


Рис. 4. Пример конфигурации памяти после выполнения нескольких операций вставки и удаления.

выбрать следующие начальные условия:

$$\texttt{BASE}[j] = \texttt{TOP}[j] = \left\lfloor \left(\frac{j-1}{n}\right)(\mathrm{L}_\infty - \mathrm{L}_0) \right\rfloor + \mathrm{L}_0 \qquad \text{для } 1 \le j \le n. \tag{13}$$

Очевидно, что, обладая определенным опытом работы с некоторой программой, можно предложить более подходящие начальные значения. Однако независимо от типа начального распределения таким образом можно избежать лишь незначительного фиксированного количества событий переполнения, причем этот эффект будет заметен только на начальных стадиях работы программы (см. упр. 17).

Другой возможный способ усовершенствования рассматриваемого метода основан на выделении при каждой переупаковке памяти объема свободного пространства, достаточного для размещения более чем одного нового элемента таблицы. Эта идея была использована Й. В. Гарвиком, который предложил полностью переупаковывать память при каждом событии переполнения на основе изменения каждого стека по окончании последней переупаковки. В его алгоритме используется дополнительный массив значений $\texttt{OLDTOP}[j]$, $1 \le j \le n$, которые равны значениям $\texttt{TOP}[j]$ сразу после предыдущего выделения памяти. В исходном состоянии таблицы выглядят так же, как и прежде, т. е. $\texttt{OLDTOP}[j] = \texttt{TOP}[j]$. Этот алгоритм состоит из следующих шагов.

Алгоритм G (*Перераспределение последовательных таблиц*). Предположим, что переполнение (OVERFLOW) произошло в стеке i согласно условиям (9). После выполнения алгоритма G либо будет исчерпана емкость памяти, либо память будет переупорядочена таким образом, что сможет быть выполнена операция $\texttt{NODE(TOP}[i]) \leftarrow \texttt{Y}$. (Обратите внимание, что значение $\texttt{TOP}[i]$ увеличено в (9) еще до применения алгоритма G.)

G1. [Инициализация.] Пусть $\texttt{SUM} \leftarrow \mathrm{L}_\infty - \mathrm{L}_0$, $\texttt{INC} \leftarrow 0$. Выполним шаг G2 для $1 \le j \le n$. (В результате SUM будет равно общему размеру доступного пространства в памяти, а INC — общему количеству последовательных увеличений размеров стеков по завершении последнего распределения памяти.) Выполнив эти действия, перейдем к шагу G3.

G2. [Сбор статистических данных.] Пусть $\texttt{SUM} \leftarrow \texttt{SUM} - (\texttt{TOP}[j] - \texttt{BASE}[j])$. Если $\texttt{TOP}[j] > \texttt{OLDTOP}[j]$, то $\texttt{D}[j] \leftarrow \texttt{TOP}[j] - \texttt{OLDTOP}[j]$ и $\texttt{INC} \leftarrow \texttt{INC} + \texttt{D}[j]$; в противном случае $\texttt{D}[j] \leftarrow 0$.

G3. [Ресурсы памяти исчерпаны?] Если $\texttt{SUM} < 0$ работа не может быть продолжена.

G4. [Вычисление показателей перераспределения памяти.] Пусть $\alpha \leftarrow 0.1 \times \texttt{SUM}/n$, $\beta \leftarrow 0.9 \times \texttt{SUM}/\texttt{INC}$. (Здесь α и β — дробные, а не целые числа, которые необходимо вычислить с заданной точностью. На следующем этапе доступное пространство будет распределено среди отдельных списков следующим образом: приблизительно 10% доступной памяти будет распределено поровну среди n стеков, а оставшиеся 90% будут разделены пропорционально увеличению размера стеков со времени предыдущего распределения памяти.)

G5. [Вычисление новых базовых адресов.] Пусть $\texttt{NEWBASE}[1] \leftarrow \texttt{BASE}[1]$ и $\sigma \leftarrow 0$; тогда для $j = 2, 3, \ldots, n$ установим такие значения: $\tau \leftarrow \sigma + \alpha + \texttt{D}[j-1]\beta$, $\texttt{NEWBASE}[j] \leftarrow \texttt{NEWBASE}[j-1] + \texttt{TOP}[j-1] - \texttt{BASE}[j-1] + \lfloor\tau\rfloor - \lfloor\sigma\rfloor$ и $\sigma \leftarrow \tau$.

G6. [Переупаковка.] Пусть $\texttt{TOP}[i] \leftarrow \texttt{TOP}[i] - 1$. (Таким образом задается истинный размер i-го стека для того, чтобы не предпринимались попытки переместить информацию за его пределами.) Выполните приведенный ниже алгоритм R, а затем установите $\texttt{TOP}[i] \leftarrow \texttt{TOP}[i] + 1$. Наконец, установите $\texttt{OLDTOP}[j] \leftarrow \texttt{TOP}[j]$ для $1 \le j \le n$. ▮

Вероятно, наиболее интересной частью всего этого алгоритма является сам процесс перераспределения, который описывается ниже. Он не совсем тривиален, поскольку одни фрагменты памяти смещаются вниз, а другие — вверх. Очевидно, что очень важно не наложить перемещаемый фрагмент памяти на другую полезную информацию, после чего она может быть полностью утрачена.

Алгоритм R (*Перемещение последовательных таблиц*). Для $1 \le j \le n$ информация, заданная с помощью адресов $\texttt{BASE}[j]$ и $\texttt{TOP}[j]$ в соответствии с указанными выше соглашениями, перемещается на новое место, задаваемое с адресами $\texttt{NEWBASE}[j]$, а сами величины $\texttt{BASE}[j]$ и $\texttt{TOP}[j]$ после этого принимают новые значения. Данный алгоритм основан на легко проверяемом условии, что перемещаемые вниз данные не должны пересекаться с любыми другими перемещаемыми вверх данными, а также с любыми неперемещаемыми данными.

R1. [Инициализация.] Установить $j \leftarrow 1$.

R2. [Поиск начала перемещения.] (Теперь все перемещаемые вниз стеки с 1- до j-го перемещены в новое место.) Будем последовательно увеличивать значение j с шагом 1 до тех пор, пока не выполнится одно из перечисленных ниже условий:
a) $\texttt{NEWBASE}[j] < \texttt{BASE}[j]$: перейти к шагу R3;
b) $j > n$: перейти к шагу R4.

R3. [Перемещение списка вниз.] Установить $\delta \leftarrow \texttt{BASE}[j] - \texttt{NEWBASE}[j]$. Установить $\texttt{CONTENTS}(\texttt{L} - \delta) \leftarrow \texttt{CONTENTS}(\texttt{L})$ для $\texttt{L} = \texttt{BASE}[j] + 1, \texttt{BASE}[j] + 2, \ldots, \texttt{TOP}[j]$. (Возможен вариант, когда значение $\texttt{BASE}[j]$ равно $\texttt{TOP}[j]$; тогда никаких действий предпринимать не следует.) Установить $\texttt{BASE}[j] \leftarrow \texttt{NEWBASE}[j]$, $\texttt{TOP}[j] \leftarrow \texttt{TOP}[j] - \delta$. Вернуться к шагу R2.

R4. [Поиск начала перемещения.] (Теперь все перемещаемые вниз стеки с j-го по n-й перемещены в нужное место.) Будем последовательно уменьшать значение j с шагом 1 до тех пор, пока не выполнится одно из перечисленных ниже условий:
a) $\texttt{NEWBASE}[j] > \texttt{BASE}[j]$: перейти к шагу R5;
b) $j = 1$: прервать выполнение алгоритма.

R5. [Перемещение списка вверх.] Установить $\delta \leftarrow$ NEWBASE[j] $-$ BASE[j]. Установить CONTENTS(L $+ \delta$) $\leftarrow$ CONTENTS(L) для L $=$ TOP[j], TOP[j] $- 1, \ldots,$ BASE[j] $+ 1$. (Как и во время выполнения шага R3, здесь возможен такой вариант, когда никаких действий предпринимать не потребуется.) Установить BASE[j] $\leftarrow$ NEWBASE[j], TOP[j] $\leftarrow$ TOP[j] $+ \delta$. Вернуться к шагу R4. ▮

Обратите внимание, что стек 1 никогда не придется перемещать. Следовательно, если известно, какой стек обладает наибольшим размером, для повышения эффективности работы программы его можно расположить первым.

В алгоритмах G и R мы намеренно предположили, что выполняются следующие условия:

$$\texttt{OLDTOP}[j] \equiv \texttt{D}[j] \equiv \texttt{NEWBASE}[j+1]$$

для $1 \le j \le n$, т. е. эти три таблицы могут совместно использовать одну и ту же область памяти, поскольку находящиеся в них значения никогда не применяются одновременно с возникновением конфликтных ситуаций.

Описанные выше алгоритмы перераспределения предназначены для работы со стеками, хотя, очевидно, их можно легко адаптировать для любых таблиц с относительными адресами, в которых текущая информация хранится между BASE[j] и TOP[j]. Для работы со списками могут применяться и другие указатели (например, FRONT[j] и REAR[j]), что позволяет манипулировать ими так же, как очередями и деками. В упр. 8 этот алгоритм подробно рассмотрен на примере очереди.

Математический анализ таких алгоритмов динамического распределения памяти чрезвычайно трудно выполнить. В приведенных ниже примерах можно найти некоторые интересные, хотя и поверхностные, результаты, поскольку в них, в основном, рассматривается общее поведение.

В качестве примера *возможного* построения теории рассмотрим случай, когда таблицы растут только за счет операций вставки. При этом игнорируются операции удаления и последующих вставок, которые компенсируют одна другую. Допустим, что каждая таблица заполняется значениями с одинаковой скоростью. Такая ситуация может моделироваться последовательностью операций вставки $a_1, a_2, \ldots, a_m$, где каждый элемент a_i — это целое число от 1 до n (представляющее вставку элемента в верхнюю часть стека a_i). Например, последовательность 1, 1, 2, 2, 1 означает две операции вставки в стек 1, две операции вставки в стек 2 и, наконец, еще одну вставку в стек 1. Допустим, что все n^m возможных реализаций $a_1, a_2, \ldots, a_m$ являются равновероятными. Каким тогда будет среднее количество операций, необходимых для перемещения одного слова из одной ячейки памяти в другую при переупаковке создаваемой таблицы? Для первого алгоритма, когда в исходном состоянии вся доступная память предоставляется n-му стеку, этот вопрос рассмотрен в упр. 9. В таком случае среднее количество необходимых операций перемещения равно

$$\frac{1}{2}\Big(1 - \frac{1}{n}\Big)\binom{m}{2}. \tag{14}$$

Таким образом, как и следовало ожидать, количество операций перемещения прямо пропорционально *квадрату* количества последовательных увеличений размера таблицы. То же самое верно и для отдельных стеков с неравновероятными реализациями (см. упр. 10).

На основе этого анализа можно сделать следующий вывод: при значительном количестве операций вставки элементов в таблицы количество перемещений может быть очень большим. Это своеобразная компенсация за возможность компактной упаковки большого количества последовательных таблиц. Для анализа алгоритма G до сих пор не предложено ни одной теоретической модели, и маловероятно, что какая-либо простая теория сможет адекватно описать характеристики реальных таблиц при таких условиях работы. Однако в упр. 18 предлагается анализ самого *неблагоприятного случая* с оценкой времени выполнения, которое будет не таким большим, если память исчерпана не полностью.

Как показывает опыт, при заполнении памяти наполовину (т. е. когда доступная область занимает половину всей памяти) с помощью алгоритма G перераспределение памяти нужно выполнить лишь в незначительной степени. Здесь важно отметить, что этот алгоритм эффективен при заполнении памяти наполовину, а при практически полном заполнении, по крайней мере, можно достоверно оценить его эффективность.

Рассмотрим, однако, более подробно случай, когда таблицы почти полностью занимают память. Тогда алгоритму R для выполнения перераспределения потребуется очень много времени. Более того, события переполнения (OVERFLOW) будут происходить все чаще и чаще непосредственно перед исчерпанием памяти. Существует весьма ограниченное количество программ, которые способны работать на грани полного исчерпания памяти и без скорого ее переполнения. Причем многим программам, которые в таком режиме работы переполняют память, вероятно, приходится тратить огромное количество времени на перераспределение памяти согласно алгоритмам G и R незадолго до полного исчерпания памяти. К сожалению, недостаточно хорошо отлаженные программы часто приводят к быстрому исчерпанию ресурсов памяти. Во избежание таких затрат можно было бы прервать выполнение алгоритма G на шаге G3, если SUM меньше значения $S_{\min}$, которое задается программистом для предотвращения выполнения избыточных операций перераспределения памяти. При наличии большого количества последовательных таблиц с изменяемыми размерами *вряд ли* сто́ит надеяться на то, что можно будет использовать доступную память на все 100% и избежать ее переполнения.

Исследование алгоритма G было продолжено Д. С. Вайсом и Д. К. Ватсоном (D. S. Wise, D. C. Watson, *BIT* **16** (1976), 442–450). А. С. Френкель предложил использовать пары стеков, которые увеличиваются по направлению друг к другу (A. S. Fraenkel, *Inf. Proc. Letters* **8** (1979), 9–10).

УПРАЖНЕНИЯ

▶ **1.** [*15*] Сколько элементов может одновременно находиться в очереди без возникновения переполнения (OVERFLOW) при выполнении с очередью операций (6, a) и (7, a)?

▶ **2.** [*22*] Обобщите метод на основе правил (6, a) и (7, a) так, чтобы он был применим для любого дека с менее чем M элементами. Иначе говоря, предложите правила выполнения двух других операций: "вывод элемента с конца" и "ввод элемента с начала".

3. [*21*] Предположим, что возможности компьютера MIX расширены следующим образом: I-поле каждой инструкции имеет вид $8\mathrm{I}_1+\mathrm{I}_2$, где $0 \le \mathrm{I}_1 < 8$, $0 \le \mathrm{I}_2 < 8$. В ассемблере используется команда OP ADDRESS,I_1:I_2 или, как сейчас, OP ADDRESS,I_2, если $\mathrm{I}_1 = 0$. Она обозначает сначала "модификацию адреса" I_1 для адреса ADDRESS, затем — выполнение

"модификации адреса" I_2 для результирующего адреса и, наконец, выполнение операции OP для нового адреса. При этом модификация адреса определяется следующим образом.

0: M = A
1: M = A + rI1
2: M = A + rI2
...
6: M = A + rI6
7: M = результирующий адрес, определенный по полям ADDRESS,I_1:I_2 в ячейке A. При этом не допускается случай, когда $I_1 = I_2 = 7$ в ячейке A. (Обоснование этого ограничения приводится в упр. 5.)

Здесь A обозначает адрес до выполнения операции, а M — результат модификации адреса. Во всех случаях результат будет неопределенным, если значение M не умещается в два байта и знаковое поле. Время выполнения увеличивается на одну единицу для каждой выполненной операции "косвенной адресации" (модификация 7).

В качестве нетривиального примера предположим, что ячейка памяти 1000 содержит NOP 1000,1:7, ячейка 1001 — NOP 1000,2, а индексные регистры 1 и 2 — 1 и 2 соответственно. В таком случае команда LDA 1000,7:2 эквивалентна команде LDA 1004, так как

$$\texttt{1000,7:2} = \texttt{(1000,1:7),2} = \texttt{(1001,7),2} = \texttt{(1000,2),2} = \texttt{1002,2} = \texttt{1004}.$$

a) Используя эти средства косвенной адресации (если это необходимо), покажите, как упростить код в правой части выражений (8), чтобы экономить по две команды при каждом обращении к таблице. Насколько эффективнее будет этот код по сравнению с кодом (8)?

b) Допустим, имеется несколько таблиц, базовые адреса которых хранятся в позициях BASE + 1, BASE + 2, BASE + 3, ... Как, используя средства косвенной адресации, можно перенести I-й элемент из J-й таблицы в регистр A за счет одной команды, предполагая, что I находится в регистре rI1, а J — в регистре rI2?

c) Каким будет результат выполнения команды ENT4 X,7, если предположить, что поле (3:3) в ячейке X равно нулю?

4. [*25*] Допустим, что возможности компьютера MIX расширены так, как в упр. 3. Покажите, как можно выполнить перечисленные ниже действия с помощью *одной команды* (со вспомогательными константами).

a) Организовать бесконечный цикл за счет непрекращающейся косвенной адресации.
b) Внести в регистр A значение LINK(LINK(x)), в котором переменная связи x хранится в поле (0:2) ячейки X, значение LINK(x) — в поле (0:2) ячейки x и т. д., при условии, что поля (3:3) в этих ячейках равны нулю.
c) Внести в регистр A значение LINK(LINK(LINK(x))) при тех же условиях, что и в п. (b).
d) Внести в регистр A содержимое ячейки rI1 + rI2 + rI3 + rI4 + rI5 + rI6.
e) Умножить на четыре текущее значение регистра rI6.

▶ **5.** [*35*] Предлагаемый в упр. 3 способ расширения возможностей компьютера MIX содержит нежелательное ограничение, которое запрещает использовать "7:7" в косвенно адресованной ячейке.

a) Приведите пример того, что без такого ограничения в компьютере MIX потребовалось бы на уровне аппаратного обеспечения реализовать большой внутренний стек трехбитовых элементов. (Даже для такого мифического компьютера, как MIX, аппаратное обеспечение может оказаться очень дорогим.)
b) Объясните, почему такой стек не нужен при использовании подобного ограничения. Иначе говоря, создайте такой алгоритм, с помощью которого аппаратное обеспечение компьютера может выполнять нужные модификации адреса без использования дополнительной емкости регистра.

c) Найдите менее строгое ограничение, чем предложенное в упр. 3 ограничение для поля 7:7, которое в некоторой степени позволило бы устранить трудности, описанные в упр. 4, (c), а также не требовало дорогостоящего аппаратного обеспечения.

6. [*10*] Определите, какая из приведенных ниже последовательностей действий может вызвать переполнение или недостаток с начальной конфигурацией памяти, показанной на рис. 4:

(a) I_1; (b) I_2; (c) I_3; (d) $I_4I_4I_4I_4I_4$; (e) $D_2D_2I_2I_2I_2$.

7. [*12*] На шаге G4 алгоритма G предусмотрена операция деления на INC. Может ли значение INC быть равным нулю когда-либо в этом месте данного алгоритма?

▸ **8.** [*26*] Объясните, как можно было бы модифицировать коды (9) и (10), а также алгоритмы переупаковки в случае, когда один или несколько списков имеют структуру очереди с циркулярным порядком обработки, аналогично правилам (6, a) и (7, a).

▸ **9.** [*M27*] Используя математическую модель, описанную почти в самом конце раздела, докажите, что среднее количество перемещений определяется формулой (14). (Обратите внимание, что для выполнения последовательности действий 1, 1, 4, 2, 3, 1, 2, 4, 2, 1 потребуется $0+0+0+1+1+3+2+0+3+6=16$ перемещений.)

10. [*M28*] Изменим математическую модель в упр. 9, предположив, что таблицы могут иметь разные размеры. Иначе говоря, предположим, что p_k — это вероятность того, что $a_j = k$ для $1 \le j \le m$, $1 \le k \le n$. Таким образом, $p_1 + p_2 + \cdots + p_n = 1$, а в предыдущем упражнении рассматривался частный случай, когда $p_k = 1/n$ для всех k. Определите среднее количество перемещений, т. е. укажите, какой будет формула (14) в этом более общем случае. Относительный порядок n списков можно изменить так, что более длинные списки будут располагаться правее (или левее). При каком относительном порядке n списков среднее количество перемещений будет минимальным для данного набора вероятностей $p_1, p_2, \ldots, p_n$?

11. [*M30*] Расширим предложенную в упр. 9 задачу следующим условием: для выполнения первых t операций вставки в стек не требуется никаких перемещений, тогда как все последующие операции вставки осуществляются так же, как и прежде. Следовательно, если $t = 2$, для приведенной в упр. 9 последовательности потребуется выполнить $0+0+0+0+0+3+0+0+3+6=12$ перемещений. Каким будет среднее количество перемещений при таком дополнительном условии? [Это условие приблизительно моделирует поведение алгоритма при наличии в исходном состоянии t доступных свободных мест в каждом стеке.]

12. [*M28*] Выгоду от использования в памяти двух таблиц, которые растут по направлению друг к другу, а не занимают отдельные независимые участки памяти, можно в определенной степени оценить количественно. Для этого сначала рассмотрим модель, использованную в упр. 9 с $n = 2$. Далее предположим, что в каждой из 2^m равновероятных последовательностей $a_1, a_2, \ldots, a_m$ содержится k_1 единиц и k_2 двоек. (Здесь k_1 и k_2 соответствуют размерам двух таблиц после полного заполнения памяти. При смежном расположении таблиц данный алгоритм можно с тем же результатом применить к $m = k_1 + k_2$ ячейкам вместо $2\max(k_1, k_2)$ ячеек при раздельном расположении таблиц.)

Каким будет среднее значение $\max(k_1, k_2)$?

13. [*HM42*] Величина $\max(k_1, k_2)$ из упр. 12 будет даже больше, если более крупные флуктуации таблиц будут вызваны не только случайными операциями *удаления*, но и случайными операциями вставки. Предположим, что прежняя модель изменена таким образом, что с вероятностью p значение последовательности a_j интерпретируется, как удаление, а не как вставка, а весь процесс продолжается до тех пор, пока $k_1 + k_2$ (общее количество используемых ячеек таблицы) не станет равным m. Причем операция удаления из пустого списка не дает никакого результата.

Например, если $m = 4$, можно показать, что будет получено следующее распределение вероятностей по окончании всего процесса:

$$\begin{array}{rccccc} (k_1,k_2) = & (0,4) & (1,3) & (2,2) & (3,1) & (4,0) \\ \text{с вероятностью} & \dfrac{1}{16-12p+4p^2}, & \dfrac{1}{4}, & \dfrac{6-6p+2p^2}{16-12p+4p^2}, & \dfrac{1}{4}, & \dfrac{1}{16-12p+4p^2}. \end{array}$$

Таким образом, при увеличении p увеличивается и разница между k_1 и k_2. Нетрудно показать, что в пределе при стремлении p к единице распределение k_1 становится практически равномерным, а предельное значение $\max(k_1,k_2)$ будет точно равно $\frac{3}{4}m+\frac{1}{4m}$[для четных m]. Такое поведение существенно отличается от рассмотренного в предыдущем примере (когда $p = 0$). Однако это не так уж и важно, поскольку при приближении p к единице время, необходимое для завершения процесса, будет быстро стремиться к бесконечности. Сформулированная в этом упражнении задача заключается в исследовании зависимости $\max(k_1,k_2)$ от p и m и определении асимптотических формул для заданных значений p (например, $p = \frac{1}{3}$), если m стремится к бесконечности. Особый интерес представляет случай, когда $p = \frac{1}{2}$.

14. [*HM43*] Обобщите результат упр. 12 для произвольного $n \geq 2$, показав, что для фиксированного n и стремящегося к бесконечности m величина

$$\frac{m!}{n^m} \sum_{\substack{k_1+k_2+\cdots+k_n=m \\ k_1,k_2,\ldots,k_n\geq 0}} \frac{\max(k_1,k_2,\ldots,k_n)}{k_1!\,k_2!\ldots k_n!}$$

имеет асимптотический вид $m/n + c_n\sqrt{m} + O(1)$. Определите константы c_2, c_3, c_4 и c_5.

15. [*40*] Используя метод Монте-Карло, промоделируйте работу алгоритма G для разных распределений вставок и удалений. Какой вывод об эффективности алгоритма G можно сделать на основании этих опытов? Сравните его производительность с производительностью другого алгоритма, в котором узлы сдвигаются вверх или вниз по одному.

16. [*20*] В этом разделе описан способ более эффективного использования общей области памяти благодаря такому расположению двух стеков в памяти, при котором они могут расти навстречу друг другу. Можно ли так расположить две *очереди* или стек и очередь, чтобы добиться такой же эффективности при использовании общей области памяти?

17. [*30*] Если σ — это произвольная последовательность вставок и удалений типа (12), пусть $s_0(\sigma)$ — количество переполнений стека, которые возникают после применения простого способа, показанного на рис. 4, к этой последовательности σ с начальными условиями наподобие (11), а $s_1(\sigma)$ — соответствующее количество переполнений по отношению к другим начальным условиям наподобие (13). Докажите, что $s_0(\sigma) \leq s_1(\sigma) + \mathtt{L}_\infty - \mathtt{L}_0$.

▸ **18.** [*M30*] Покажите, что общее время выполнения любой последовательности m вставок и/или удалений согласно алгоритмам G и R имеет порядок $O(m + n\sum_{k=1}^{m} \alpha_k/(1-\alpha_k))$, где α_k — доля памяти, занятая во время последней переупаковки памяти до k-й операции. $\alpha_k = 0$ до первой переупаковки памяти. (Следовательно, если память никогда не заполняется более чем на 90%, для выполнения каждой операции потребуется не больше $O(n)$ тактов независимо от общего размера памяти.) Предполагается, что $\mathtt{L}_\infty - \mathtt{L}_0 \geq n^2$.

▸ **19.** [*16*] (*Нуль-индексирование.*) Опытным программистам известно, что в общем случае для индексирования элементов линейного списка лучше использовать форму `X[0]`, `X[1]`, ..., `X[`$n-1$`]` вместо традиционной формы `X[1]`, `X[2]`, ..., `X[`n`]`. Тогда, например, базовый адрес $\mathtt{L}_0$ в (1) будет указывать на наименьшую ячейку массива.

Измените методы вставки и удаления (2, а), (3, а), (6, а) и (7, а) для стеков и очередей так, чтобы они соответствовали этому соглашению. Иначе говоря, измените их так, чтобы все элементы списка находились в массиве `X[0]`, `X[1]`, ..., `X[M` $-$ `1]`, а не в массиве `X[1]`, `X[2]`, ..., `X[M]`.

2.2.3. Связанное распределение

Вместо того чтобы хранить линейный список в последовательных ячейках памяти, можно использовать более гибкую схему, в соответствии с которой каждый узел содержит связь со следующим узлом списка.

Последовательное распределение

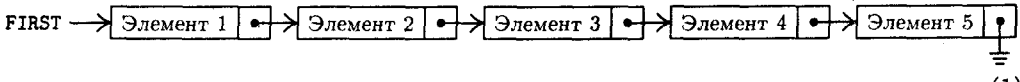
Адрес	Содержимое
$L_0 + c$:	Элемент 1
$L_0 + 2c$:	Элемент 2
$L_0 + 3c$:	Элемент 3
$L_0 + 4c$:	Элемент 4
$L_0 + 5c$:	Элемент 5

Связанное распределение

Адрес	Содержимое	
A:	Элемент 1	B
B:	Элемент 2	C
C:	Элемент 3	D
D:	Элемент 4	E
E:	Элемент 5	Λ

Здесь A, B, C, D и E — это произвольные ячейки памяти, а Λ — пустая связь (см. раздел 2.1). В программе с такой таблицей с последовательным распределением элементов для обозначения ее длины (5 элементов) потребуется либо дополнительная переменная или константа, либо специальный код в последнем, 5-м, элементе или в следующей за ним ячейке памяти. В программе со связанным распределением необходимо использовать переменную или константу связи, которая будет указывать на адрес A, причем все другие элементы списка могут быть найдены с помощью этого адреса.

Как упоминалось выше, в разделе 2.1, связи часто схематически изображают в виде стрелок, так как их действительное расположение в памяти обычно не имеет никакого значения. В таком случае описанная выше таблица при связанном распределении памяти будет выглядеть следующим образом.



(1)

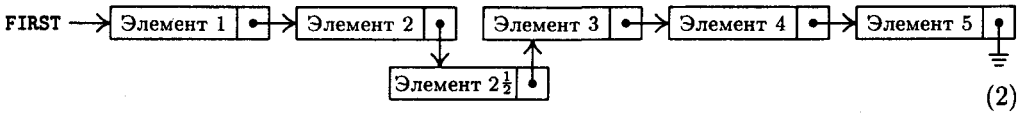
Здесь FIRST — это переменная связи, которая указывает на первый узел списка.

Сравнив два основных типа хранилищ, можно сделать несколько очевидных выводов.

1) При организации связанного распределения в памяти потребуется выделить дополнительное пространство для размещения связей. В некоторых ситуациях этот фактор может оказаться доминирующим. Но часто бывает так, что хранимая в узле информация все равно не занимает все слово целиком, поэтому место для связи всегда найдется. Кроме того, во многих случаях несколько элементов комбинируются в одном узле, а потому связь может использоваться сразу для нескольких элементов данных (см. упр. 2.5–2). Однако важнее то, что с помощью связанного распределения памяти *выигрыш* можно получить неявно за счет частичного перекрытия таблиц, совместно использующих некоторые области памяти. Часто последовательное распределение не так рационально, как связанное, поскольку для его эффективной работы необходимо очень большое количество дополнительных вакантных ячеек памяти. Так, в конце предыдущего раздела уже объяснялось, почему подобные системы становятся крайне неэффективными при плотном заполнении памяти.

2) Операция удаления элемента в связанном списке выполняется проще. Например, для удаления элемента 3 необходимо только изменить связь с ним в элементе 2. А при последовательном выделении памяти такое удаление в общем случае подразумевает перемещение значительной части списка в другие ячейки.

3) При использовании схемы связанного распределения памяти проще выполняется и вставка элемента в середину списка. Например, для вставки элемента $2\frac{1}{2}$ в список (1) потребуется изменить только две связи.



(2)

А при работе с длинной таблицей с последовательным выделением памяти для вставки нужно будет выполнить гораздо больше действий, для чего потребуется чрезвычайно много времени.

4) Обращения к произвольным элементам списка гораздо быстрее осуществляются при последовательном выделении памяти. Для доступа к k-му элементу списка, где k — некоторая переменная, в случае последовательного распределения памяти потребуется фиксированное время, но для доступа к нужному элементу в случае связанного распределения памяти потребуется выполнить k итераций. Таким образом, полезность связанного распределения памяти основывается на том, что в большинстве приложений доступ к элементам списка выполняется в последовательном, а не произвольном порядке. Для доступа к элементам в середине или в конце списка можно создать дополнительную переменную связь или целый список переменных связи, которые указывают на нужные позиции списка.

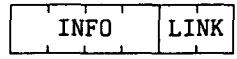
5) В схеме связанного распределения памяти проще организовать объединение двух списков или разбиение списка на два других списка, которые могут увеличиваться независимо.

6) Схема связанного распределения памяти прекрасно подходит для организации более сложных структур, чем простые линейные списки. Например, ее можно применить для создания переменного количества списков переменного размера. Причем каждый узел одного из них может быть началом другого списка, к тому же узлы могут быть связаны между собой несколькими способами и образовывать другие списки и т. д.

7) Такие простые операции, как последовательная обработка элементов списка, на многих компьютерах выполняется немного быстрее. Для компьютера MIX операторы `INC1 c` и `LD1 0,1(LINK)` отличаются только одним тактом, но на многих компьютерах не реализована возможность загрузки индексного регистра из индексированной ячейки. Если элементы связанного списка принадлежат разным страницам устройства хранения большого объема, то операция доступа к ним может выполняться значительно дольше.

Таким образом, технология связанного распределения памяти позволяет обойти ограничения, накладываемые последовательной природой компьютерной памяти. При выполнении одних операций она способствует достижению гораздо более высокой эффективности, а при выполнении других, наоборот, снижает ее. Обычно совершенно ясно, какая технология лучше всего подходит в конкретной ситуации. Поэтому часто в одной и той же программе используют списки разных типов.

В следующих нескольких примерах предположим для удобства, что узел состоит из одного слова, которое содержит два поля — INFO и LINK:



(3)

При использовании связанного распределения памяти обычно подразумевается, что существует некоторый механизм поиска свободного места для нового узла при вставке новых данных в список. Это обычно выполняется с помощью специального *списка свободного пространства*. Здесь он будет называться списком AVAIL (или стеком AVAIL, поскольку он обычно обрабатывается, как стек магазинного типа, т. е. по принципу "последним вошел — первым вышел" (last-in-first-out)). Набор всех неиспользуемых в данный момент узлов связан в список, который устроен так же, как и все другие списки. Переменная связи AVAIL ссылается на самый верхний элемент этого списка. Таким образом, если необходимо присвоить переменной связи X адрес нового узла, а сам узел зарезервировать для дальнейшего использования, то можно выполнить следующие действия:

$$\texttt{X} \leftarrow \texttt{AVAIL}, \qquad \texttt{AVAIL} \leftarrow \texttt{LINK(AVAIL)}. \tag{4}$$

При этом из стека AVAIL будет удален самый верхний элемент, а X будет указывать на только что удаленный узел. *Операция* (4) *в дальнейшем будет использоваться настолько часто, что для нее лучше ввести специальное обозначение:* $\texttt{X} \Leftarrow \texttt{AVAIL}$ будет означать, что X указывает на новый узел.

Для удаления ненужного узла операцию (4) можно обратить:

$$\texttt{LINK(X)} \leftarrow \texttt{AVAIL}, \qquad \texttt{AVAIL} \leftarrow \texttt{X}. \tag{5}$$

При этом узел с адресом, указанным с помощью X, будет возвращен в список свободного пространства, а вся операция (5) будет обозначаться как $\texttt{AVAIL} \Leftarrow \texttt{X}$.

При обсуждении правил работы со стеком AVAIL были опущены некоторые важные подробности. Например, ничего не было сказано о его создании после запуска программы. Очевидно, что стек может быть создан следующим образом: (a) за счет связывания всех узлов, которые предполагается использовать для организации связанной памяти, (b) путем присвоения адреса первого из этих узлов переменной связи AVAIL и (c) в результате присвоения значения Λ связи последнего узла. Набор всех выделенных таким образом узлов называется *пулом* (*storage pool*).

Однако самой важной особенностью новой структуры является проверка переполнения. В операции (4) не предусмотрена проверка ситуации, когда в памяти исчерпано все свободное пространство. На самом деле для этого операция $\texttt{X} \Leftarrow \texttt{AVAIL}$ должна быть определена иначе:

$$\begin{aligned}&\text{Если } \texttt{AVAIL} = \Lambda, \text{ то } \texttt{OVERFLOW};\\ &\qquad\qquad \text{в противном случае } \texttt{X} \leftarrow \texttt{AVAIL}, \ \texttt{AVAIL} \leftarrow \texttt{LINK(AVAIL)}.\end{aligned} \tag{6}$$

Возможность переполнения должна быть предусмотрена всегда! В данной ситуации переполнение (OVERFLOW) означает либо прекращение работы программы (к сожалению), либо запуск процедуры "сборки мусора" (garbage collection), которая предпримет попытку поиска свободного пространства. Сборка мусора более подробно описывается в разделе 2.3.5.

Для управления стеком AVAIL можно использовать другую технологию, поскольку часто заранее неизвестно, какой объем памяти потребуется для пула. Для этого можно применить последовательную таблицу переменного размера, которая может сосуществовать в памяти вместе со связанными таблицами. В таком случае нужно позаботиться о том, чтобы связанная область памяти не занимала больше места, чем необходимо. Предположим, что связанная область памяти размещается в ячейках с возрастающими адресами, начиная с L_0, и что эта область никогда не выходит за пределы величины SEQMIN (которая представляет собой текущую нижнюю границу для последовательной таблицы). В таком случае в результате применения новой переменной POOLMAX происходит следующее.

a) Выполняется установка AVAIL $\leftarrow \Lambda$ и POOLMAX $\leftarrow L_0$.

b) Операция X $\Leftarrow$ AVAIL теперь выглядит так:

Если AVAIL $\neq \Lambda$, то X $\leftarrow$ AVAIL, AVAIL $\leftarrow$ LINK(AVAIL);
иначе X $\leftarrow$ POOLMAX и POOLMAX $\leftarrow$ X $+ c$, где c — размер узла; (7)
OVERFLOW имеет место, если POOLMAX $>$ SEQMIN.

c) Другие части программы при попытках уменьшить значение SEQMIN подают сигнал о переполнении (OVERFLOW), если SEQMIN $<$ POOLMAX.

d) Операция AVAIL $\Leftarrow$ X остается неизменной, т. е. такой же, как и в (5).

Эта идея на самом деле представляет собой просто вставку специальной процедуры восстановления, используемой при переполнении (OVERFLOW) в условии (6). Суммарный результат заключается в поддержании пула наименьшего размера. Многие склонны использовать эту идею, даже если *все* списки находятся в пуле (так что величина SEQMIN остается постоянной), поскольку с ее помощью можно избежать очень неэкономной операции исходного связывания всех ячеек и к тому же упростить отладку. Кроме того, последовательный список можно разместить снизу, а пул — сверху, используя POOLMIN и SEQMAX вместо POOLMAX и SEQMIN.

Следовательно, в пуле свободных узлов довольно просто и эффективно можно выполнять поиск и возврат свободных узлов. Описанные методы предоставляют возможность получить материал для использования в связанных таблицах. Эти рассуждения основаны на неявном предположении о том, что все узлы имеют одинаковый размер, c, хотя большое значение имеют также случаи с узлами разных размеров. Их рассмотрение будет продолжено в разделе 2.5. Теперь опишем несколько наиболее общих операций со списками при работе со стеками и очередями.

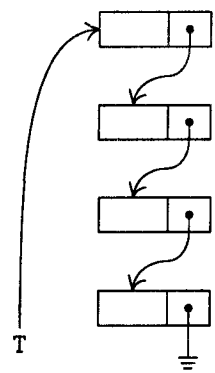


Рис. 5. Связанный стек.

Простейшим типом связанного списка является стек. На рис. 5 показан типичный стек с указателем T на верхний элемент стека. Когда стек пуст, этот указатель имеет значение Λ.

Теперь ясно, какие операции следует выполнить для вставки ("проталкивания") нового элемента данных Y сверху такого стека, используя вспомогательный указатель P:

$$\text{P} \Leftarrow \text{AVAIL}, \qquad \text{INFO(P)} \leftarrow \text{Y}, \qquad \text{LINK(P)} \leftarrow \text{T}, \qquad \text{T} \leftarrow \text{P}. \tag{8}$$

И наоборот: для "выталкивания" верхнего элемента из стека и передачи его данных узлу Y следует выполнить такие операции:

Если T = Λ, то UNDERFLOW;

в противном случае P ← T, T ← LINK(P), Y ← INFO(P), AVAIL ⇐ P. (9)

Их полезно сравнить с аналогичными механизмами работы последовательно организованных стеков, (2, а) и (3, а) в разделе 2.2.2. Читателю следует тщательно изучить операции (8) и (9), поскольку они обладают исключительной важностью.

До изучения очередей найдем наиболее удобное выражение операций со стеками в программах для компьютера MIX. Программа вставки элемента, где P ≡ rI1, может выглядеть так.

```
INFO EQU  0:3          Определение поля INFO.
LINK EQU  4:5          Определение поля LINK.
     LD1  AVAIL        P ← AVAIL.          }
     J1Z  OVERFLOW     Верно ли AVAIL = Λ? } P ⇐ AVAIL.
     LDA  0,1(LINK)                        }
     STA  AVAIL        AVAIL ← LINK(P).    }
     LDA  Y                                              (10)
     STA  0,1(INFO)    INFO(P) ← Y.
     LDA  T
     STA  0,1(LINK)    LINK(P) ← T.
     ST1  T            T ← P.              ∎
```

Для ее выполнения потребуется 17 тактов, по сравнению с 12 тактами для аналогичной операции в последовательной таблице (хотя для обработки события переполнения (OVERFLOW) при последовательном распределении памяти в большинстве случаев необходимо гораздо больше времени). В этой программе, как и в других программах данной главы, OVERFLOW обозначает либо завершение процедуры, либо подпрограмму поиска свободного пространства и возвращения в позицию rJ − 2.

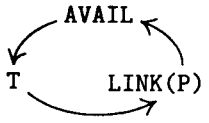
Программа удаления элемента также очень проста.

```
LD1  T            P ← T.
J1Z  UNDERFLOW    Верно ли, что T = Λ?
LDA  0,1(LINK)
STA  T            T ← LINK(P).
LDA  0,1(INFO)                                          (11)
STA  Y            Y ← INFO(P).
LDA  AVAIL                            }
STA  0,1(LINK)    LINK(P) ← AVAIL.    } AVAIL ⇐ P.
ST1  AVAIL        AVAIL ← P.          }  ∎
```

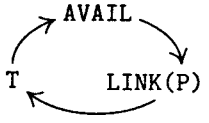
Интересно, что для выполнения этих операций необходимо осуществить циклическую перестановку трех ссылок. Допустим, что перед выполнением операции вставки элемента указатель P равен AVAIL. Если P ≠ Λ, то после выполнения этой операции

значение AVAIL стало равным предыдущему значению LINK(P),
значение LINK(P) стало равным предыдущему значению T и
значение T стало равным предыдущему значению AVAIL.

Таким образом, процесс вставки (за исключением присвоения INFO(P) ← Y) является циклической перестановкой



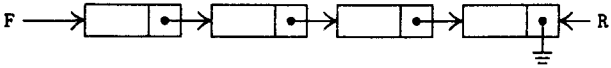
Аналогично в случае удаления элемента, если до выполнения этой операции P имеет значение T и предполагается, что P ≠ Λ, получим Y ← INFO(P) и



Факт цикличности этой перестановки на самом деле здесь не так уж и важен, поскольку *любая* перестановка трех элементов со смещением каждого элемента является циклической. Гораздо важнее то, что в данных операциях изменяются в точности три связи.

Хотя алгоритмы вставки и удаления (8) и (9) созданы для стеков, их можно применять и в более широком смысле для вставки и удаления в *любом* линейном списке. Допустим, требуется вставить элемент непосредственно перед узлом, на который указывает переменная связи T. Тогда вставка элемента $2\frac{1}{2}$ в приведенном выше списке (2) может быть выполнена с помощью операции (8), где T = LINK(LINK(FIRST)).

Связанное распределение памяти особенно удобно применять для очередей. Нетрудно заметить, что связи должны быть направлены от начала очереди к ее концу, а при удалении начального узла должен существовать механизм прямого указания нового начального узла. Здесь указатели начального и конечного узлов очереди будут обозначены символами F и R соответственно:



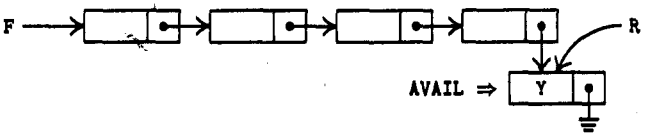
(12)

За исключением указателя R, эта схема практически идентична схеме на рис. 5 на с. 298.

Всякий раз при проектировании списка важно предусмотреть все возможные ситуации, особенно случай, когда список пуст. Одной из наиболее распространенных ошибок, возникающих при работе со связанным выделением памяти, является неадекватная обработка пустых списков. Другая распространенная ошибка может быть вызвана тем, что после выполнения манипуляций структурой программист забывает изменить некоторые связи. Для того чтобы избежать ошибок первого типа, следует всегда внимательно проверять "граничные условия". А чтобы не совершать ошибок второго типа, полезно создавать диаграммы состояния до и после выполнения некоторой операции, а затем, сравнивая их, обнаруживать те связи, которые потребуется изменить.

Проиллюстрируем замечания из предыдущего абзаца, применив их к очередям. Сначала рассмотрим операцию вставки: если диаграмма (12) соответствует состоянию до вставки, то диаграмма состояния после вставки элемента данных с конца

очереди будет выглядеть так:

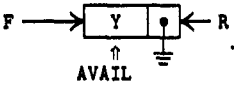


(13)

(Согласно использованным здесь обозначениям новый узел получен из списка AVAIL.) Сравнивая (12) и (13), можно предложить следующие действия, которые необходимо выполнить для вставки элемента данных Y с конца очереди:

$$\texttt{P} \Leftarrow \texttt{AVAIL}, \quad \texttt{INFO(P)} \leftarrow \texttt{Y}, \quad \texttt{LINK(P)} \leftarrow \Lambda, \quad \texttt{LINK(R)} \leftarrow \texttt{P}, \quad \texttt{R} \leftarrow \texttt{P}. \tag{14}$$

А теперь рассмотрим "граничное" состояние, когда очередь пуста. В этом случае состояние до вставки еще потребуется определить, а состояние "после" выглядит так:



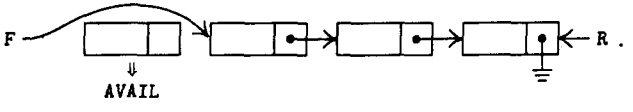
(15)

Было бы желательно снова использовать операции (14), даже если для вставки в пустую очередь придется изменить *оба* указателя, F и R, а не только R. Нетрудно обнаружить, что операции (14) можно будет использовать для этого, если R = LOC(F), когда очередь пуста, *при условии, что* F ≡ LINK(LOC(F)). Иначе говоря, для реализации этой идеи значение переменной F должно *храниться* в *поле* LINK *ее же ячейки*. Для максимально эффективной проверки граничных условий при работе с пустой очередью предположим, что F = Λ. Следовательно,

пустая очередь задается указателями $F = \Lambda$ *и* $R = LOC(F)$.

Выполнив операции (14) при этих условиях, получим (15).

Операция удаления элемента данных из очереди может быть получена аналогичным образом. Если (12) описывает состояние очереди до удаления элемента, то ее состояние после удаления будет таким:



(16)

При работе с граничными условиями следует убедиться в том, что операция удаления корректно выполняется как до, так и после опустошения очереди. Исходя из этих рассуждений, можно предложить следующий способ удаления элемента данных из очереди в общем случае:

Если F = Λ, то UNDERFLOW;
в противном случае P ← F, F ← LINK(P), Y ← INFO(P), AVAIL ⇐ P, (17)
а если F = Λ, то R ← LOC(F).

Обратите внимание, что R необходимо будет изменить, если очередь станет пустой. Это именно тот тип "граничного условия", который всегда следует отслеживать.

Изложенный выше метод — вовсе не единственный способ представления очередей в виде линейно связанного списка. Например, в упр. 30 описывается в некоторой

степени даже более естественный альтернативный способ, а в конце этой главы приводятся другие методы. Действительно, ни одна из перечисленных выше операций не может рассматриваться как единственный возможный способ реализации того или иного действия. Они представляют собой лишь примеры основных приемов работы со связанными списками. Читателю, который не обладает богатым опытом работы с такими методами, прежде чем продолжить чтение этой книги, будет полезно еще раз перечитать приведенный выше материал данного раздела.

До сих пор в настоящей главе обсуждались способы выполнения некоторых операций с таблицами, но форма обсуждения всегда была достаточно "абстрактной" в том смысле, что никогда не демонстрировались реальные программы, в которых использовались бы данные методы. Человек не станет изучать абстрактные модели некоторой проблемы, если его не заинтересовали примеры. Рассмотренные до сих пор операции, т. е. вставка и удаление данных в списках с изменяемыми размерами, а также использование таблиц в виде стеков или очередей имеют настолько широкое применение, что читатель наверняка еще не раз встретится с ними и сможет оценить их важность. Однако в остальной части данной главы мы покинем область абстрактных рассуждений и приступим к изучению важных практических примеров использования этих методов.

Первый пример такой практической задачи называется *топологической сортировкой* (*topological sorting*), которую часто приходится выполнять при проектировании сетей, в так называемых PERT-диаграммах, и даже в лингвистике. Действительно, она может быть очень полезной всегда, когда приходится выполнять *частичное упорядочение* (*partial ordering*). Частичным упорядочением некоторого множества S называется такое отношение между элементами множества S, которое может быть обозначено символом "$\preceq$" и обладает следующими свойствами для любых x, y и z (причем необязательно разных) элементов множества S.

i) Если $x \preceq y$ и $y \preceq z$, то $x \preceq z$. (Транзитивность.)

ii) Если $x \preceq y$ и $y \preceq x$, то $x = y$. (Антисимметричность.)

iii) $x \preceq x$. (Рефлексивность.)

$x \preceq y$ можно трактовать так: "x предшествует или равно y". Если $x \preceq y$ и $x \neq y$, будем обозначать это отношение через $x \prec y$ и трактовать его как "x предшествует y". Легко видеть, что из (i)–(iii) можно получить следующее.

i′) Если $x \prec y$ и $y \prec z$, то $x \prec z$.(Транзитивность.)

ii′) Если $x \prec y$, то $y \not\prec x$. (Антисимметричность.)

iii′) $x \not\prec x$. (Нерефлексивность.)

Отношение, обозначенное через $y \not\prec x$, трактуется так: "y не предшествует x". Если сначала определить отношение $\prec$ со свойствами (i′)–(iii′), то описанный выше процесс можно выполнить в обратной последовательности, т. е. определить отношение $x \preceq y$, как такое, для которого $x \prec y$ или $x = y$. Тогда для него должны выполняться условия (i)–(iii). Следовательно, свойства (i)–(iii) и (i′)–(iii′) можно рассматривать как определения частичного упорядочения. Обратите внимание на то, что свойство (ii′) на самом деле является следствием свойств (i′) и (iii′), однако свойство (ii) не следует из свойств (i) и (iii).

Частичное упорядочение довольно часто встречается не только в математике, но и в повседневной жизни. Примерами его использования в математике являются отношение $x \leq y$ между действительными числами x и y, отношение $x \subseteq y$ между множествами, отношение $x \backslash y$ (x делит y) между положительными целыми числами. В PERT-диаграммах S — это набор заданий, которые следует выполнить, а отношение $x \prec y$ означает "x должно быть выполнено раньше y".

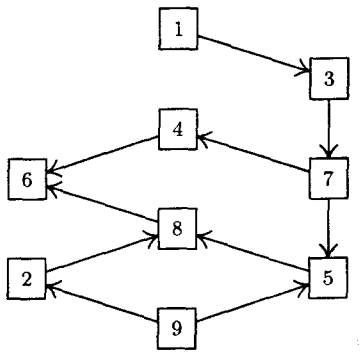


Рис. 6. Частичное упорядочение.

Вполне естественно было бы предположить, что S является конечным множеством, поскольку оно используется в компьютере. Частичное упорядочение конечного множества всегда можно представить в виде диаграммы, изображенной на рис. 6, на которой объекты изображены маленькими квадратиками, а отношение между ними — стрелками между этими квадратиками. В таком случае $x \prec y$ означает, что от квадратика с ярлыком x к квадратику с ярлыком y проходит некоторый путь, указанный стрелками. Свойство (ii) частичного упорядочения означает, что в такой схеме *не существует замкнутых петель* (т. е. не существует путей, замкнутых на себя). Например, если в схеме на рис. 6 провести стрелку от квадратика 4 к квадратику 1, то для них нельзя будет установить отношение частичного упорядочения.

Задача топологической сортировки заключается в *установлении частичного порядка среди объектов, упорядоченных в линейном порядке*, т. е. в таком расположении объектов линейной последовательности $a_1 a_2 \ldots a_n$, чтобы для любых $a_j \prec a_k$ выполнялось условие $j < k$. Графически это означает, что квадратики следует расположить на одной линии так, чтобы все стрелки были направлены вправо (рис. 7). Такое переупорядочение не всегда возможно, хотя совершенно ясно, что его нельзя выполнить при наличии замкнутых петель. Следовательно, искомый алгоритм интересен не только тем, что он позволяет выполнить очень полезную операцию, но и тем, что он доказывает, что данная операция *возможна* для любого частичного упорядочения.

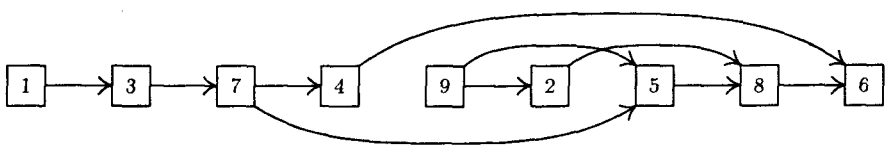


Рис. 7. Отношение упорядочения, показанного на рис. 6, после выполнения топологической сортировки.

В качестве примера топологической сортировки представим себе большой словарь с определениями технических терминов. Запишем, что $w_2 \prec w_1$, если определение термина w_1 прямо или косвенно зависит от определения термина w_2. Это отношение является частичным упорядочением при условии, что не существует никаких "циклических определений". Тогда задача топологической сортировки заключается в *поиске такого способа упорядочения терминов в этом словаре, чтобы никакой термин не использовался до того, как он будет определен.* Аналогичные проблемы возникают при создании программ для обработки объявлений в некоторых ассемблерных и компилируемых языках, а также при создании руководства пользователя с описанием языка программирования или при написании учебников об информационных структурах.

Существует очень простой способ выполнения топологической сортировки. Сортировку следует начать с объекта, которому не предшествует никакой другой объект данного множества. Этот объект удаляется из исходного множества S и располагается первым в итоговой последовательности. Таким образом итоговая последовательность частично упорядочивается и процесс снова повторяется до тех пор, пока все множество не будет рассортировано. Например, на рис. 6 сортировку можно было бы начать, удалив элемент 1 или 9, затем (после удаления элемента 1) — удалив элемент 3 и т. д. Этот алгоритм может оказаться бесполезным только тогда, когда найдется такое непустое частично упорядоченное множество, каждому элементу которого предшествует другой элемент. Но если каждый элемент имеет предшественника, то можно построить последовательность произвольной длины $b_1, b_2, b_3, \ldots$, в которой $b_{j+1} \prec b_j$. Так как S является конечным множеством, для некоторого $j < k$ должно выполняться условие $b_j = b_k$. Но тогда предполагается, что выполняется условие $b_k \preceq b_{j+1}$, а это противоречит свойству (ii).

Для эффективной реализации рассмотренного процесса на компьютере необходимо организовать выполнение перечисленных выше действий, т. е. поиск объектов без предшественников, а также их удаление из множества. На окончательный вид алгоритма также повлияют заданные характеристики операций ввода и вывода. В общем случае программа должна уметь воспринимать символьные имена объектов и обрабатывать гигантские множества, размер которых может даже превышать размер оперативной памяти компьютера. Однако обсуждаемые здесь основные идеи могут легко затеряться среди таких усложнений. Поэтому более подробно методы эффективной обработки символьных данных описываются в главе 6, а задачу управления большими сетями читателю предлагается обдумать самостоятельно в качестве интересного упражнения.

Далее предположим, что сортируемые объекты пронумерованы от 1 до n в произвольном порядке. Вводная часть программы будет находиться на накопителе на магнитной ленте 1. Каждая магнитная запись содержит 50 пар чисел, где пара (j, k) означает, что объект j предшествует объекту k. Однако первая пара выглядит как $(0, n)$, где n — это количество объектов. Ввод завершается парой $(0, 0)$. Предположим, что n и все пары помещаются в памяти компьютера, а потому необязательно проверять наличие свободного места в памяти во время ввода этих данных. Выводиться объекты будут уже в отсортированном порядке на накопитель на магнитной ленте 2, причем за ними будет следовать значение 0.

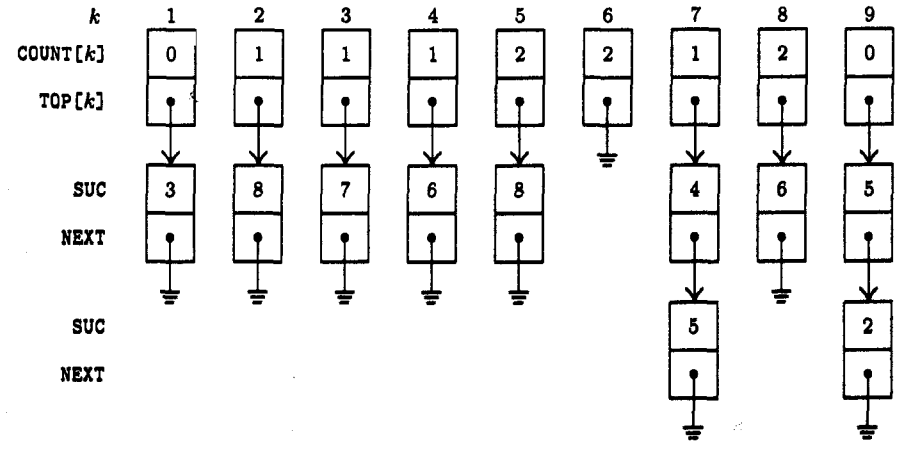


Рис. 8. Компьютерное представление показанной на рис. 6 последовательности элементов с отношениями (18).

Например, поток ввода может содержать следующие объекты с такими отношениями между ними:

$$9 \prec 2,\ 3 \prec 7,\ 7 \prec 5,\ 5 \prec 8,\ 8 \prec 6,\ 4 \prec 6,\ 1 \prec 3,\ 7 \prec 4,\ 9 \prec 5,\ 2 \prec 8. \qquad (18)$$

Здесь необязательно приводить какие-то дополнительные отношения для характеристики искомого частичного упорядочения. Например, дополнительные отношения типа $9 \prec 8$ (которые можно вывести из отношений $9 \prec 5$ и $5 \prec 8$) можно без какого-либо влияния на конечный результат опустить или добавить во входной поток данных. В общем случае необходимо указать только те отношения, которые соответствуют стрелкам на диаграмме, показанной на рис. 6.

В приведенном ниже алгоритме используется последовательно упорядоченная таблица `X[1]`, `X[2]`, ..., `X[`n`]`, в которой каждый узел `X[`k`]` имеет вид

+	0	COUNT[k]	TOP[k]

.

Здесь `COUNT[`k`]` обозначает *количество прямых предшественников* объекта k (т. е. количество отношений $j \prec k$ во входном потоке данных), а `TOP[`k`]` — связь с началом *списка прямых потомков* объекта k. Последний список содержит элементы в формате

+	0	SUC	NEXT

,

где `SUC` — прямой потомок объекта k и `NEXT` — следующий элемент списка. Для демонстрации этих условных обозначений на рис. 8 схематически показано содержимое памяти, соответствующее входному потоку данных (18).

Используя такую организацию памяти, нетрудно создать искомый алгоритм. Для этого следует вывести узлы, в которых значение поля `COUNT` равно нулю, и на единицу уменьшить значения полей `COUNT` для всех потомков таких узлов. Уловка здесь заключается в том, чтобы избежать "поиска" узлов, в которых значение поля `COUNT` равно нулю, и это можно выполнить за счет организации очереди, содержащей такие узлы. Связи для этой очереди хранятся в поле `COUNT`, которое уже выполнило

свое назначение. Для ясности в приведенном ниже алгоритме обозначение QLINK[k] используется вместо COUNT[k], когда это поле больше не применяется для хранения значений счетчика.

Алгоритм Т (*Топологическая сортировка*). В этом алгоритме в качестве входного потока используется последовательность отношений $j \prec k$, обозначающих, что объект j предшествует объекту k в некотором частичном упорядочении при условии, что $1 \le j, k \le n$. Выходной поток состоит из множества n объектов, расположенных в линейном порядке. При этом используются следующие внутренние таблицы: QLINK[0], COUNT[1] = QLINK[1], COUNT[2] = QLINK[2], ..., COUNT[n] = QLINK[n]; TOP[1], TOP[2], ..., TOP[n]; пул с одним узлом для каждого отношения входного потока и с указанными выше полями SUC и NEXT; переменная связи P, которая используется для ссылки на узлы в пуле; целочисленные переменные F и R, применяемые для ссылок на начало и конец очереди, связи которых находятся в таблице QLINK; и, наконец, переменная N, позволяющая подсчитать количество объектов, которые необходимо поместить в выходной поток.

Т1. [Инициализация.] Ввести значение n. Установить COUNT[k] $\leftarrow 0$ и TOP[k] $\leftarrow \Lambda$ для $1 \le k \le n$. Установить N $\leftarrow n$.

Т2. [Следующее отношение.] Получить отношение $j \prec k$ из входного потока. Если входной поток исчерпан, перейти к шагу Т4.

Т3. [Запись отношения.] Увеличить на единицу значение COUNT[k]. Установить

$$\texttt{P} \Leftarrow \texttt{AVAIL},\ \texttt{SUC(P)} \leftarrow k,\ \texttt{NEXT(P)} \leftarrow \texttt{TOP}[j],\ \texttt{TOP}[j] \leftarrow \texttt{P}.$$

(Это операция (8).) Вернуться к шагу Т2.

Т4. [Поиск нулей.] (В этом месте завершается ввод, и входной поток (18) теперь имеет понятный для компьютера вид, представленный на рис. 8. Следующий шаг заключается в инициализации очереди выходного потока, который связан с помощью поля QLINK.) Установить R $\leftarrow 0$ и QLINK[0] $\leftarrow 0$. Для $1 \le k \le n$ проверить значение COUNT[k] и, если оно равно нулю, установить QLINK[R] $\leftarrow k$ и R $\leftarrow k$. После выполнения этого действия для всех k установить F $\leftarrow$ QLINK[0] (теперь оно будет содержать первое значение k, для которого COUNT[k] равно нулю).

Т5. [Вывод начала очереди.] Вывести значение F. Если F $= 0$, перейти к шагу Т8, в противном случае установить N $\leftarrow$ N $- 1$ и установить P $\leftarrow$ TOP[F]. (Так как таблицы QLINK и COUNT перекрываются, получим QLINK[R] $= 0$. Следовательно, условие F $= 0$ выполняется, когда очередь пуста.)

Т6. [Удаление отношений.] Если P $= \Lambda$, перейти к шагу Т7. В противном случае уменьшить на единицу значение COUNT[SUC(P)], а если оно уже равно нулю, установить QLINK[R] $\leftarrow$ SUC(P) и R $\leftarrow$ SUC(P). Установить P $\leftarrow$ NEXT(P) и повторить этот шаг. (Таким образом удаляем из системы все отношения вида F $\prec k$ для некоторого k и располагаем новые узлы в очереди, когда все их предшественники уже выведены.)

Т7. [Удаление из очереди.] Установить F $\leftarrow$ QLINK[F] и вернуться к шагу Т5.

Т8. [Конец процесса.] Прекращение работы алгоритма. Если N $= 0$, получим в результате искомое "топологическое упорядочение" пронумерованных объектов,

за которыми следует нуль. В противном случае N пронумерованных объектов не будут выведены из-за того, что они образуют замкнутую петлю, а это нарушает гипотезу о частичном упорядочении. (В упр. 23 рассмотрен алгоритм печати содержимого такой петли.) ▮

Читателю наверняка будет полезно попробовать вручную применить этот алгоритм для входного потока (18). Алгоритм Т демонстрирует прекрасное взаимодействие между методами организации последовательной и связанной памяти. Последовательная память используется для основной таблицы X[1], ..., X[n], которая содержит объекты COUNT[k] и TOP[k], поскольку на шаге Т3 предстоит ссылаться на "произвольно выбранные" части этой таблицы. (Однако, если входной поток является символьным, для более быстрого поиска следует использовать другой тип таблицы, который описывается в главе 6.) Связанная память применяется для таблиц, в которых "один объект непосредственно следует за другим" (immediate successors), так как во входном потоке эти объекты никак не упорядочены. Очередь ожидающих вывода узлов хранится внутри последовательной таблицы, связывая, таким образом, узлы в порядке их вывода. Для этого связывания вместо адресов используется индекс таблицы. Иначе говоря, если X[k] — начало очереди, получим F = k вместо F = LOC(X[k]). Операции обработки очереди, используемые на шагах Т4, Т6 и Т7, не идентичны операциям (14) и (17), так как в данном случае используются преимущества особых свойств очереди. Во время выполнения этой части алгоритма никакие узлы не придется создавать или возвращать в свободное пространство памяти.

При программировании алгоритма Т на языке ассемблера компьютера MIX следует учесть несколько интересных особенностей. Так как в данном алгоритме в таблице не предпринимается никаких удалений (поскольку не требуется освобождать память для ее последующего использования), операция P ⇐ AVAIL может быть выполнена чрезвычайно просто, т. е. так, как показано ниже, в строках 19 и 32. В такой ситуации не потребуется содержать связанный пул памяти, а новые узлы можно выбирать последовательно. Ввод и вывод данных в программе предполагается выполнять с магнитной ленты согласно упомянутым выше соглашениям, *но для упрощения кода в данном случае опущена буферизация.* Читатель вряд ли столкнется с какими-либо трудностями при чтении этого кода, поскольку код полностью соответствует алгоритму Т. Здесь также показано, насколько эффективно может быть организовано использование индексных регистров , что является важным аспектом обработки связанной памяти.

Программа Т (*Топологическая сортировка*). В этой программе (рис. 9) нужно учесть следующие равенства: rI6 ≡ N, rI5 ≡ указатель буфера, rI4 ≡ k, rI3 ≡ j и R, rI2 ≡ AVAIL и P, rI1 ≡ F, TOP[j] ≡ X + j(4:5), COUNT[k] ≡ QLINK[k] ≡ X + k(2:3).

```
01  *  БУФЕРИЗАЦИЯ И ОПРЕДЕЛЕНИЕ ПОЛЕЙ
02  COUNT   EQU  2:3          Определение символьных
03  QLINK   EQU  2:3              имен полей.
04  TOP     EQU  4:5
05  SUC     EQU  2:3
06  NEXT    EQU  4:5
07  TAPEIN  EQU  1            Ввод с ленты 1.
08  TAPEOUT EQU  2            Вывод на ленту 2.
```

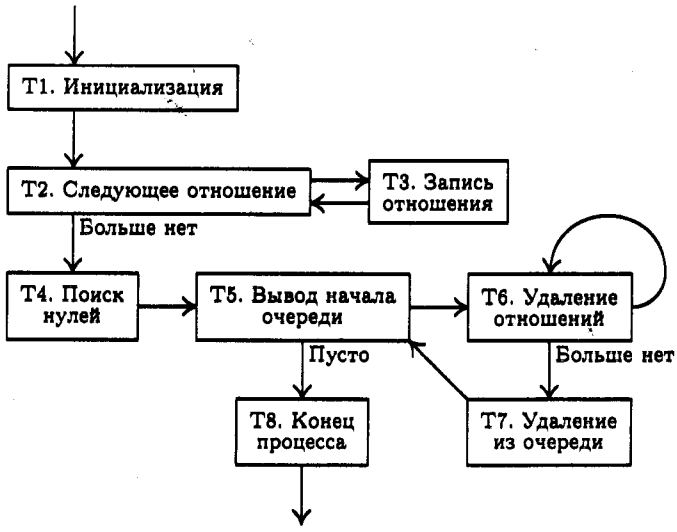


Рис. 9. Топологическая сортировка.

```
09 BUFFER  ORIG *+100                    Буферная область.
10         CON  -1                       Метка конца буфера.
11 *  ФАЗА ВВОДА
12 TOPSORT IN   BUFFER(TAPEIN)     1     T1. Инициализация. Считать первый
13         JBUS *(TAPEIN)                    блок ленты; дождаться завершения.
14 1H      LD6  BUFFER+1           1     N ← n.
15         ENT4 0,6                1
16         STZ  X,4                n+1   Установить COUNT[k] ← 0 и TOP[k] ← Λ
17         DEC4 1                  n+1         для 0 ≤ k ≤ n.
18         J4NN *-2                n+1   (Учесть, что QLINK[0] ← 0 на шаге T4.)
19         ENT2 X,6                1     Свободная область начинается после X[n].
20         ENT5 BUFFER+2           1     Приготовиться к чтению первой пары (j,k).
21 2H      LD3  0,5                m+b   T2. Следующее отношение.
22         J3P  3F                 m+b   Верно ли, что j > 0?
23         J3Z  4F                 b     Входной поток исчерпан?
24         IN   BUFFER(TAPEIN)     b-1   Метка конца замечена; считать другой
25         JBUS *(TAPEIN)                    блок с ленты, дождаться завершения.
26         ENT5 BUFFER             b-1   Переустановить указатель буфера.
27         JMP  2B                 b-1
28 3H      LD4  1,5                m     T3. Запись отношения.
29         LDA  X,4(COUNT)         m     COUNT[k]
30         INCA 1                  m       +1
31         STA  X,4(COUNT)         m           → COUNT[k].
32         INC2 1                  m     AVAIL ← AVAIL + 1.
33         LDA  X,3(TOP)           m     TOP[j]
34         STA  0,2(NEXT)          m        → NEXT(P).
35         ST4  0,2(SUC)           m     k → SUC(P).
36         ST2  X,3(TOP)           m     P → TOP[j].
37         INC5 2                  m     Увеличить указатель буфера.
38         JMP  2B                 m
```

```
39  4H      IOC   0(TAPEIN)           1      Перемотать ленту ввода.
40          ENT4  0,6                 1      T4. Поиск нулей.  k ← n.
41          ENT5  -100                1      Переустановить указатель буфера вывода.
42          ENT3  0                   1      R ← 0.
43  4H      LDA   X,4(COUNT)          n      Проверить значение COUNT[k].
44          JAP   *+3                 n      Не равно ли оно нулю?
45          ST4   X,3(QLINK)          a      QLINK[R] ← k.
46          ENT3  0,4                 a      R ← k.
47          DEC4  1                   n
48          J4P   4B                  n      n ≥ k ≥ 1.
49  *   ФАЗА СОРТИРОВКИ
50          LD1   X(QLINK)            1      F ← QLINK[0].
51  5H      JBUS  *(TAPEOUT)                 T5. Вывод начала очереди.
52          ST1   BUFFER+100,5        n+1    Сохранить F в области буфера.
53          J1Z   8F                  n+1    Не равно ли нулю значение F?
54          INC5  1                   n      Увеличить указатель буфера.
55          J5N   *+3                 n      Проверить, заполнен ли буфер.
56          OUT   BUFFER(TAPEOUT)     c-1    Если дополнен, переписать блок на ленту.
57          ENT5  -100                c-1    Переустановить указатель буфера.
58          DEC6  1                   n      N ← N − 1.
59          LD2   X,1(TOP)            n      P ← TOP[F].
60          J2Z   7F                  n      T6. Удаление отношений.
61  6H      LD4   0,2(SUC)            m      rI4 ← SUC(P).
62          LDA   X,4(COUNT)          m      COUNT[rI4]
63          DECA  1                   m        −1
64          STA   X,4(COUNT)          m           → COUNT[rI4].
65          JAP   *+3                 m      Достигнут ли нуль?
66          ST4   X,3(QLINK)          n-a    Если достигнут, установить QLINK[R] ← rI4.
67          ENT3  0,4                 n-a    R ← rI4.
68          LD2   0,2(NEXT)           m      P ← NEXT(P).
69          J2P   6B                  m      Если P ≠ Λ, повторить.
70  7H      LD1   X,1(QLINK)          n      T7. Удаление из очереди.
71          JMP   5B                  n      F ← QLINK(F), перейти к шагу T5.
72  8H      OUT   BUFFER(TAPEOUT)     1      T8. Конец процесса.
73          IOC   0(TAPEOUT)          1      Вывести последний блок и перемотать ленту.
74          HLT   0,6                 1      Останов с выводом значения N на консоль.
75  X       END   TOPSORT                    Начало области таблицы.  ▮
```

Анализ алгоритма T достаточно просто можно выполнить с помощью закона Кирхгофа. Используя этот закон, время выполнения можно приблизительно оценить с помощью формулы c_1m+c_2n, где m — количество введенных отношений, n — количество объектов, а c_1 и c_2 — константы. Более быстрый алгоритм для решения этой задачи просто невозможно себе представить! Что касается программы T, то для нее можно получить точную оценку времени выполнения, используя следующие параметры: a = количество объектов, не имеющих предшественника, b = количество блоков на входе, $= \lceil (m+2)/50 \rceil$, и c = количество блоков на выходе $= \lceil (n+1)/100 \rceil$. Если не принимать во внимание продолжительность операций ввода-вывода, общее время выполнения в таком случае будет равно всего лишь $(32m+24n+7b+2c+16)u$.

Метод топологической сортировки, аналогичный алгоритму T (но без важной особенности связанных списков), впервые был опубликован А. Б. Каном (A. B. Kahn,

CACM **5** (1962), 558–562), а доказательство того, что топологическая сортировка частичного упорядочения всегда возможна, было впервые опубликовано Э. Шпильрайном (E. Szpilrajn, *Fundamenta Mathematica* **16** (1930), 386–389). Он доказал этот результат как для бесконечных, так и для конечных множеств, причем упомянул, что он уже был известен некоторым из его коллег.

Несмотря на столь высокую эффективность алгоритма T в разделе 7.4.1 будет рассмотрен еще более эффективный алгоритм топологической сортировки.

УПРАЖНЕНИЯ

▸ **1.** [*10*] В операции (9) для "выталкивания" элемента из стека предусмотрена возможность обработки события недостатка (UNDERFLOW). Почему тогда в операции (8) для "проталкивания" элемента в стек не предусмотрена возможность обработки события переполнения (OVERFLOW)?

2. [*22*] Напишите подпрограмму "общего назначения" для компьютера MIX, которая выполняла бы операцию вставки (10). Эта подпрограмма должна удовлетворять следующим требованиям (так же, как и в разделе 1.4.1).

Условия вызова: `JMP INSERT` Переход к подпрограмме.
`NOP T` Адрес указательной переменной.
Условия ввода: rA = информация, которая должна быть введена в поле INFO нового узла.
Условия вывода: Стек, указатель которого является переменной связи T и имеет сверху новый узел; rI1 = T; rI2, rI3 изменяются.

3. [*22*] Напишите подпрограмму "общего назначения" для компьютера MIX, которая выполняла бы операцию удаления (11). Эта подпрограмма должна удовлетворять следующим требованиям.

Условия вызова: `JMP DELETE` Переход к подпрограмме.
`NOP T` Адрес указательной переменной.
`JMP UNDERFLOW` Первый выход, если происходит событие UNDERFLOW.
Условия ввода: Не определены.
Условия вывода: Если стек, указателем которого является переменная связи T, пуст, срабатывает первый выход, в противном случае удаляется верхний узел стека и выход переносится к третьей позиции после JMP DELETE. В последнем случае rI1 = T и в rA находится содержимое поля INFO удаленного узла. В *любом* случае rI2 и rI3 используются этой подпрограммой.

4. [*22*] Программа (10) основана на операции P ⇐ AVAIL, определенной действиями (6). Покажите, как можно создать такую подпрограмму обработки событий переполнения (OVERFLOW), чтобы без *каких-либо* изменений в коде (10) в операции P ⇐ AVAIL использовалось ограничение SEQMIN, указанное в (7). В общем случае эта подпрограмма не должна изменять содержимое регистров, за исключением регистра rJ и, возможно, индикатора сравнения. Она должна иметь выход в позиции rJ − 2 вместо обычной позиции rJ.

▸ **5.** [*24*] Операции (14) и (17) эквивалентны операциям с очередью. Покажите, как можно было бы определить операцию "вставка элемента с начала очереди" для получения набора всех действий, выполняемых для дека с ограниченным вводом. Как в таком случае определить операцию "удаление с конца" (для получения дека общего типа)?

6. [*21*] В операции (14) было установлено LINK(P) ← Λ, тогда как следующая операция вставки элемента в конце очереди изменит значение того же самого поля связи. Покажите, как можно было бы избежать установки LINK(P) в (14) при внесении изменений в проверку условия F = Λ в (17).

▸ **7.** [*23*] Предложите алгоритм "обращения" связанного линейного списка наподобие (1), т. е. такого изменения связей, чтобы его элементы расположились в обратном порядке. [Если, например, обратить список (1), то в нем указатель FIRST будет связан с узлом, содержащим элемент 5, а этот узел будет связан с узлом, содержащим элемент 4, и т. д.] Допустим, что узлы имеют вид (3).

8. [*24*] Напишите программу для компьютера MIX, чтобы выполнить упр. 7, оптимизируя скорость ее выполнения.

9. [*20*] Какое из приведенных ниже отношений является частичным упорядочением некоторого множества S? [*Замечание.* Если ниже определено отношение $x \prec y$, то задача заключается в определении отношения $x \preceq y \equiv (x \prec y$ или $x = y)$ и выяснении, является ли $\preceq$ частичным упорядочением.] (a) $S =$ множество всех рациональных чисел; $x \prec y$ означает, что $x > y$. (b) $S =$ множество всех людей; $x \prec y$ означает, что x является предком y. (c) $S =$ множество всех целых чисел; $x \preceq y$ означает, что x кратно y (т. е. $x \bmod y = 0$). (d) $S =$ множество всех доказанных в этой книге математических результатов; $x \prec y$ означает, что доказательство y зависит от истинности x. (e) $S =$ множество всех положительных целых чисел; $x \preceq y$ означает, что $x + y$ четно. (f) $S =$ множество подпрограмм; $x \prec y$ означает, что x вызывает y, т. е. подпрограмма y может быть вызвана во время работы подпрограммы x, но рекурсия при этом не допускается.

10. [*M21*] При условии, что $\subset$ является отношением, которое удовлетворяет свойствам (i) и (ii) частичного упорядочения, докажите, что отношение $\preceq$, определенное правилом "$x \preceq y$ тогда и только тогда, когда $x = y$ или $x \subset y$", обладает всеми тремя свойствами частичного упорядочения.

▸ **11.** [*24*] Результат топологической сортировки не всегда полностью определен, поскольку может существовать несколько способов упорядочения узлов и удовлетворения условий топологического порядка. Найдите все возможные способы топологического упорядочения узлов, показанных на рис. 6.

12. [*M20*] Существует 2^n подмножеств множества n элементов, и эти подмножества частично упорядочены с помощью отношения включения множества. Предложите два способа упорядочения данных подмножеств в топологическом порядке.

13. [*M48*] Сколько существует способов упорядочения в топологическом порядке 2^n подмножеств, описанных в упр. 12? (Дайте ответ в виде функции от n.)

14. [*M21*] *Линейным упорядочением* (*linear ordering*) или *полным упорядочением* (*total ordering*) множества S называется частичное упорядочение, которое удовлетворяет дополнительному условию "сравнимости".

(iv) Для любых двух элементов x, y множества S верно либо $x \preceq y$, либо $y \preceq x$.

Докажите непосредственно на основе этого определения, что топологическая сортировка может быть получена в единственном варианте тогда и только тогда, когда отношение $\preceq$ является линейным упорядочением. (Предположим, что множество S конечно.)

15. [*M25*] Покажите, что для любого частичного упорядочения конечного множества S существует *единственное* множество неприводимых отношений, которое характеризует это упорядочение, например таких, как отношения (18) и отношения, показанные на рис. 6. Верно ли это утверждение, если S является бесконечным множеством?

16. [*M22*] Для любого заданного частичного упорядочения множества $S = \{x_1, \ldots, x_n\}$ можно построить его *матрицу инцидентности* (*incidence matrix*) (a_{ij}), где $a_{ij} = 1$, если $x_i \preceq x_j$, и $a_{ij} = 0$ в противном случае. Покажите, что существует такая перестановка строк и столбцов этой матрицы, при которой все ее элементы, расположенные ниже диагонали, будут равны нулю.

▸ **17.** [*21*] Каким будет результат выполнения алгоритма T для исходного набора отношений (18)?

18. [*20*] Какой смысл имеют значения `QLINK[0]`, `QLINK[1]`, ..., `QLINK[`n`]` после завершения алгоритма T (если они вообще его имеют)?

19. [*18*] В алгоритме T на шаге T5 проверяется начальный элемент очереди, но этот элемент не удаляется из очереди до шага T7. Что произойдет, если установить `F` $\leftarrow$ `QLINK[F]` в конце шага T5, а не на шаге T7?

▸ **20.** [*24*] `F`, `R` и таблица `QLINK` используются в алгоритме T для организации очереди с узлами, в которых поле `COUNT` уже стало равным нулю, но их отношения с наследниками еще не были удалены. Можно ли для этого вместо очереди использовать стек? Если можно, то сравните полученный алгоритм с алгоритмом T.

21. [*21*] Сможет ли алгоритм T правильно выполнять топологическую сортировку при многократном повторении отношения $j \prec k$ во входном потоке? Что произойдет, если входной поток будет содержать некоторое отношение в виде $j \prec j$?

22. [*23*] В программе T предполагается, что на входной магнитной ленте содержится корректная информация, но в программе общего назначения всегда следует предусматривать тщательную проверку входного потока для обнаружения описок и попыток "самоуничтожения" программы. Например, если одно из входных отношений является отрицательным для некоторого k, программа T может ошибочно изменить одну из своих команд во время записи в `X[`k`]`. Предложите такой способ изменения программы T, который позволил бы избежать подобных сбоев.

▸ **23.** [*27*] Если алгоритм топологической сортировки не может продолжить работу из-за обнаружения замкнутой петли во входном потоке (см. шаг T8), вряд ли стоит останавливать его работу только для вывода сообщения "Возникла петля". Лучше распечатать одну из петель с отображением части входного потока, который привел к ошибке. Расширьте алгоритм T так, чтобы в нем была предусмотрена дополнительная возможность распечатки петли в случае необходимости. [*Указание.* В этом разделе приводится доказательство существования петли, если `N` > 0 на шаге T8, на основании которого можно создать данный алгоритм.]

24. [*24*] Реализуйте расширения алгоритма T из упр. 23 в коде программы T.

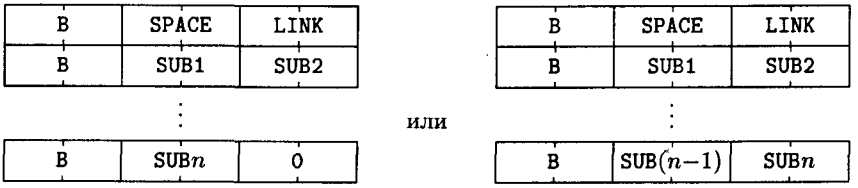
25. [*47*] Напишите максимально эффективный алгоритм для выполнения топологической сортировки для очень больших множеств S, которые содержат гораздо больше узлов, чем может поместиться в памяти компьютера. Предположим, что операции ввода, вывода и временного хранения данных выполняются с помощью накопителя на магнитной ленте. [*Указание.* При обычной сортировке входных данных предполагается, что все отношения узла располагаются рядом. Что в таком случае можно было бы предпринять? В частности, необходимо предусмотреть самый неблагоприятный случай, когда задано сильно перемешанное линейное упорядочение. В упр. 24, во введении к главе 5, описывается вариант решения этой задачи с помощью $O(\log n)^2$ итераций.]

26. [*29*] (*Распределение памяти для подпрограмм.*) Допустим, что на магнитной ленте хранится основная библиотека подпрограмм в перемещаемом виде, который широко использовался в компьютерах 60-х годов. Процедуре загрузки нужно определить величину перемещения для каждой применяемой подпрограммы, чтобы для загрузки каждой необходимой программы потребовалось выполнить только один проход по всем данным. Проблема заключается в том, что для работы одних подпрограмм необходимо загрузить в память другие подпрограммы. При этом редко используемые подпрограммы (которые обычно располагаются в конце ленты) могут вызывать часто используемые подпрограммы

(которые обычно располагаются в начале ленты), и потому нужно знать обо всех необходимых подпрограммах до выполнения прохода по всем данным на ленте.

Один из способов решения этой проблемы заключается в хранении "каталога ленты" в памяти. Процедура загрузки обладает правом доступа к двум таблицам.

a) Каталог ленты. Эта таблица состоит из узлов переменной длины следующего вида:



Здесь SPACE — количество слов памяти, необходимых для данной подпрограммы; LINK — ссылка на элемент каталога для подпрограммы, которая находится сразу за данной подпрограммой; SUB1, SUB2, ..., SUBn ($n \geq 0$) — связи с элементами каталога для других подпрограмм, которые необходимы для работы данной подпрограммы; B = 0 для всех слов, за исключением последнего, B = −1 для последнего слова в узле. Адрес элемента каталога для первой подпрограммы на магнитной ленте с библиотекой подпрограмм задается переменной связи FIRST.

b) Список подпрограмм, непосредственно указанных загружаемой программой. Он хранится в последовательных позициях X[1], X[2], ..., X[N], где N ≥ 0 — переменная, которая известна процедуре загрузки. Каждый элемент этого списка является связью с элементом каталога для соответствующей подпрограммы.

Процедуре загрузки также известно количество перемещений (MLOC), которые необходимо выполнить для загрузки первой подпрограммы.

В качестве небольшого примера рассмотрим следующую конфигурацию.

Каталог ленты

	B	SPACE	LINK
1000:	0	20	1005
1001:	−1	1002	0
1002:	−1	30	1010
1003:	0	200	1007
1004:	−1	1000	1006
1005:	−1	100	1003
1006:	−1	60	1000
1007:	0	200	0
1008:	0	1005	1002
1009:	−1	1006	0
1010:	−1	20	1006

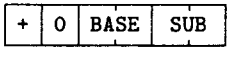
Необходимые подпрограммы

X[1] = 1003
X[2] = 1010

N = 2
FIRST = 1002
MLOC = 2400

Каталог ленты в данном случае показывает, что в указанном порядке на ленте хранятся подпрограммы 1002, 1010, 1006, 1000, 1005, 1003 и 1007. Подпрограмма 1007 занимает 200 позиций, и при ее работе предполагается использовать подпрограммы 1005, 1002 и 1006; и т. д. Для загрузки этой подпрограммы потребуются подпрограммы 1003 и 1010, которые будут размещены в ячейках по адресам ≥ 2400. В свою очередь, для этих подпрограмм потребуется загрузить также подпрограммы 1000, 1006 и 1002.

Блок распределения памяти подпрограммы должен изменить таблицу X так, чтобы каждый элемент X[1], X[2], ... принял следующую форму:



(за *исключением последнего элемента, который описывается ниже*), где SUB — загружаемая подпрограмма и BASE — величина перемещения. Данные элементы должны быть расположены в том же порядке, в котором они располагаются на ленте. Ниже показано одно из возможных решений задачи из приведенного выше примера.

	BASE	SUB		BASE	SUB
X[1]:	2400	1002	X[4]:	2510	1000
X[2]:	2430	1010	X[5]:	2530	1003
X[3]:	2450	1006	X[6]:	2730	0

Последний элемент содержит значение адреса первой неиспользуемой ячейки памяти.

(*Очевидно, что это не единственный способ работы с библиотекой подпрограмм. Соответствующая ему организация библиотеки в значительной степени зависит от типа используемого компьютера и выполняемых приложений. При работе с мощными современными компьютерами для организации библиотеки подпрограмм потребуется совершенно другой подход. Тем не менее рассмотренный выше метод является прекрасным примером использования интересных приемов работы с последовательными и связанными данными.*)

Назначение этого упражнения состоит в создании алгоритма решения поставленной задачи. Созданный для этого блок распределения памяти может при подготовке решения преобразовывать каталог ленты произвольным образом, так как каталог ленты может считываться снова и снова блоком распределения памяти при следующем распределении, причем каталог ленты не будет востребован никакими другими частями процедуры загрузки.

27. [*25*] Напишите программу для компьютера MIX, реализующую алгоритм из упр. 26.

28. [*40*] Приведенная ниже конструкция демонстрирует способ "решения" достаточно общей задачи — игры для двух лиц, например шахматы, ним или другие более простые игры. Рассмотрим конечное множество узлов, каждый из которых представляет собой возможное состояние игры. Существует несколько шагов (или ни одного), которые позволяют преобразовать каждое состояние в некоторое другое. Назовем состояние x предшественником состояния y (а y — наследником состояния x), если существует такой шаг, который переводит состояние x в состояние y. Состояния без наследников называются *проигрышем* или *выигрышем*. Игрок, делающий ход с переходом в состояние x, является соперником игрока, который делает ход с переходом от состояния x к состояниям-наследникам.

Имея такую конфигурацию состояний, полный набор выигрышных состояний (в которых может выиграть игрок, делающий следующий ход) и полный набор проигрышных состояний (в которых обязательно проигрывает игрок, делающий следующий ход) можно вычислить, повторяя следующую операцию до тех пор, пока не будет достигнуто неизменное состояние. Обозначим состояние "проигрышным", если все состояния-наследники являются "выигрышными", и обозначим состояние "выигрышным", если хотя бы одно состояние-наследник является "проигрышным".

После бесконечного повторения этой операции некоторые состояния могут быть никак не обозначены. Это значит, что игрок, делающий следующий ход из такого состояния, не может выиграть, но его нельзя победить.

Такая процедура вычисления полного набора выигрышных и проигрышных состояний может быть сформулирована для компьютера в виде алгоритма, подобного алгоритму T. Для этого можно сохранять для каждого состояния количество его состояний-наследников, которые не отмечены как "выигрышные", а также список всех его предшественников.

Назначение этого упражнения состоит в создании конкретного алгоритма, который лишь приблизительно описан выше, и в его применении для некоторых интересных игр, не содержащих слишком большого количества возможных состояний [подобных "военной

игре": É. Lucas, *Récréations Mathématiques* **3** (Paris, 1893), 105–116; E. R. Berlekamp, J. H. Conway, and R. K. Guy, *Winning Ways* **2** (Academic Press, 1982), Chapter 21].

▸ **29.** [*21*] (a) Предложите алгоритм "удаления" всего списка (1) с размещением всех его узлов в стеке AVAIL, если известно только значение FIRST. При этом алгоритм должен иметь оптимизированную скорость выполнения. (b) Решите задачу из п. (a) для списка (12) при условии, что известны значения F и R.

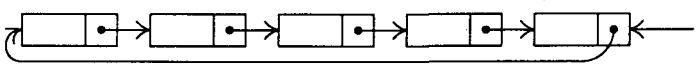
30. [*17*] Предположим, что очереди выглядят так, как (12), но с пустой очередью, представленной значением $F = \Lambda$ и *неопределенным* значением R. Как в таком случае следует изменить операции вставки и удаления, представленные правилами (14) и (17)?

2.2.4. Циклические списки

Немного изменив способ связывания, можно ввести новый метод, альтернативный предложенному в предыдущем разделе.

Циклически связанный список (*circularly linked list*) (или короче — циклический список) — это связанный список, в котором последний узел связан с первым узлом, а не с Λ. Поэтому всегда существует возможность доступа к любому элементу списка, начиная с произвольно выбранного элемента. К тому же повышается степень симметрии структуры списка, и при работе со списком можно не заботиться о том, где находится последний или первый узел.

Типичная схема циклического списка выглядит так:



(1)

Предположим, что узел содержит два поля, INFO и LINK, как и в предыдущем разделе. Ссылочная переменная PTR указывает на крайний справа узел списка; при этом в поле LINK(PTR) будет содержаться адрес крайнего слева узла. Наиболее важными являются следующие простейшие операции.

a) Вставка элемента Y слева: P ⇐ AVAIL, INFO(P) ← Y, LINK(P) ← LINK(PTR), LINK(PTR) ← P.
b) Вставка элемента Y справа: вставить Y слева, а затем — PTR ← P.
c) Передать в Y содержимое левого узла и удалить этот узел из списка: P ← LINK(PTR), Y ← INFO(P), LINK(PTR) ← LINK(P), AVAIL ⇐ P.

Операция (b), на первый взгляд, выглядит несколько необычно. Операция PTR ← LINK(PTR) на самом деле перемещает крайний слева узел в схеме (1), и это довольно легко можно представить, если рассмотреть список как замкнутое кольцо, а не прямую линию со связанными концами.

Внимательный читатель заметит, что в операциях (a)–(c) допущена серьезная ошибка. Какая? *Ответ*: не предусмотрена возможность *опустошения* списка. Если, например, операция (c) применяется пять раз по отношению к списку (1), получится, что PTR указывает на узел в списке AVAIL, а это может привести к серьезным осложнениям. Например, попробуем применить операцию (c) еще раз! Если предположить, что указатель PTR равен Λ в случае полного опустошения списка, эту ошибку можно устранить, вставив дополнительные команды "если PTR = Λ; то PTR ← LINK(P) ← P; в противном случае ..." после команды "INFO(P) ← Y" в (a), предвосхитив операцию (c) проверкой "если PTR = Λ, то UNDERFLOW" и завершив операцию (c) командой "если PTR = P, то PTR ← Λ".

Обратите внимание, что операции (a)–(c) представляют собой действия, выполняемые с деком с ограниченным выводом, который рассматривался в разделе 2.2.1. Следовательно, циклический список может использоваться либо как стек, либо как очередь. Операции (a) и (c) представляют функциональную часть стека, а операции (b) и (c) — очереди. Эти операции не так очевидны, как их аналоги из предыдущего раздела, в котором показано, что операции (a)–(c) могут выполняться с линейными списками с помощью указателей F и R.

Для циклических списков гораздо эффективнее выполняются другие важные операции. Например, очень просто выполняется операция удаления списка, т. е. размещения всего циклического списка в стеке AVAIL:

$$\text{Если PTR} \neq \Lambda, \text{ то AVAIL} \leftrightarrow \text{LINK(PTR)}. \tag{2}$$

[Напомним, что операция "↔" обозначает обмен наподобие P ← AVAIL, AVAIL ← LINK(PTR), LINK(PTR) ← P.] Очевидно, что операция (2) будет корректно выполняться, если PTR указывает *на произвольный элемент* циклического списка. После чего, конечно, следует установить PTR ← Λ.

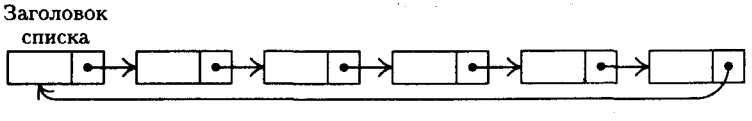
Используя этот же метод, если PTR_1 и PTR_2 указывают на отдельные циклические списки L_1 и L_2 соответственно, можно вставить список L_2 целиком справа в список L_1:

$$\begin{aligned}&\text{Если } \text{PTR}_2 \neq \Lambda, \text{ то}\\ &\quad(\text{если } \text{PTR}_1 \neq \Lambda, \text{ то } \text{LINK}(\text{PTR}_1) \leftrightarrow \text{LINK}(\text{PTR}_2);\\ &\quad\text{установить } \text{PTR}_1 \leftarrow \text{PTR}_2,\ \text{PTR}_2 \leftarrow \Lambda).\end{aligned} \tag{3}$$

Другим примером легко выполнимой операции является расщепление самыми разными способами одного циклического списка на два подсписка. Эти операции соответствуют конкатенации (сцеплению) и деконкатенации (разбиению) строк.

Таким образом, циклический список можно использовать не только для представления исходно циклических структур, но и для линейных структур. Циклический список с одним указателем на конечный узел эквивалентен линейному списку с двумя указателями на начало и конец. Естественный вопрос в данном случае формулируется так: "Как, имея дело с циклически симметричной структурой, найти конец списка?". Ведь для обозначения конца не существует никакого специального объекта типа Λ! Ответ заключается в том, что при обработке всего списка, перемещаясь от одного узла к другому, прекратить обработку следует по достижении стартовой позиции (при условии, конечно же, что такая позиция существует в данном списке).

Альтернативным решением только что поставленной задачи было бы использование в каждом циклическом списке в качестве конечной позиции особого легко распознаваемого узла. Такой особый узел называется *заголовком списка* (*list head*). Как можно будет вскоре убедиться, в приложениях часто очень удобно придерживаться такого правила: каждый циклический список должен содержать один узел, который является заголовком этого списка. Одним из преимуществ подобной структуры является то, что такой циклический список никогда не опустошается. При использовании заголовка списка схема (1) будет выглядеть так:



(4)

Обращение к списку (4) обычно выполняется с помощью заголовка списка, который часто располагается в некоторой фиксированной ячейке памяти. Недостатком применения структуры с заголовком списка является то, что в нем нет никакого указателя на правый конец, поэтому при работе с таким списком придется пожертвовать описанной выше операцией (b).

Схему (4) можно сравнить со схемой 2.2.3–(1), приведенной в начале предыдущего раздела, в которой связь с "элементом 5" теперь указывает на LOC(FIRST), а не на Λ. Переменная FIRST рассматривается как ссылка внутри узла, а именно — как ссылка, которая находится внутри NODE(LOC(FIRST)). Принципиальным отличием между (4) и 2.2.3–(1) является то, что схема (4) предоставляет возможность (хотя и необязательно эффективную) получения доступа к любому элементу списка из любого элемента списка.

В качестве примера использования циклических списков рассмотрим *арифметические действия над полиномами* от переменных x, y и z с целыми коэффициентами. Для решения многих проблем часто приходится вместо чисел манипулировать многочленами. Например, умножив

$$(x^4 + 2x^3y + 3x^2y^2 + 4xy^3 + 5y^4) \qquad \text{на} \qquad (x^2 - 2xy + y^2),$$

получим

$$(x^6 - 6xy^5 + 5y^6).$$

Для этого вполне естественно было бы применить связанное распределение, поскольку размер многочленов может непредсказуемо возрасти и может возникнуть необходимость хранить в памяти сразу несколько таких многочленов.

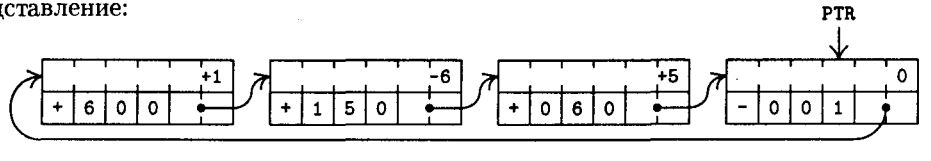
Рассмотрим здесь только две операции: сложение и умножение. Предположим, что многочлен представлен в виде списка, каждый узел которого обозначает ненулевой член многочлена, а сам узел состоит из двух слов и имеет следующий вид:

COEF				
±	A	B	C	LINK

. (5)

Здесь COEF является коэффициентом члена $x^{\mathrm{A}}y^{\mathrm{B}}z^{\mathrm{C}}$. Допустим, что коэффициенты и степени никогда не выходят за рамки диапазона, заданного в используемом формате, а потому проверять соответствие диапазону необязательно.

Обозначение ABC будет использоваться для обозначения полей ± A B C узла (5) как единого целого. Знак ABC, а именно — знак второго слова в узле (5), всегда положителен, за исключением *особого узла* в конце каждого многочлена, для которого ABC $= -1$ и COEF $= 0$. Такой особый узел очень удобно использовать по аналогии с заголовком списка, так как он выполняет функции признака конца и это позволяет решить проблему опустошения списка (что соответствует многочлену 0). Узлы списка всегда располагаются в *порядке убывания* поля ABC, если следовать в направлении заданных связей, но особый узел (у которого ABC $= -1$) связан с наибольшим узлом ABC. Например, многочлен $x^6 - 6xy^5 + 5y^6$ будет иметь такое представление:



Алгоритм А (*Сложение многочленов*). Этот алгоритм складывает многочлен (P) с многочленом (Q) при условии, что P и Q — указатели описанного выше вида. Список P останется неизменным, а список Q будет содержать сумму многочленов. По завершении алгоритма ссылочным переменным P и Q возвращаются их прежние значения. Кроме того, в алгоритме используются вспомогательные переменные Q1 и Q2.

А1. [Инициализация.] Установить P ← LINK(P), Q1 ← Q, Q ← LINK(Q). (Теперь P и Q указывают на первые члены многочленов. На протяжении почти всего этого алгоритма переменная Q1 будет на один шаг отставать от переменной Q в том смысле, что Q = LINK(Q1).)

А2. [ABC(P):ABC(Q).] Если ABC(P) < ABC(Q), установить Q1 ← Q и Q ← LINK(Q) и повторить этот шаг. Если ABC(P) = ABC(Q), перейти к шагу А3. Если ABC(P) > ABC(Q), перейти к шагу А5.

А3. [Сложение коэффициентов.] (Найдены члены с одинаковыми степенями.) Если ABC(P) < 0, выполнение алгоритма прекращается. В противном случае установить COEF(Q) ← COEF(Q) + COEF(P). Теперь, если COEF(Q) = 0, перейти к шагу А4; в противном случае установить P ← LINK(P), Q1 ← Q, Q ← LINK(Q) и перейти к шагу А2. (Любопытно, что последние операции идентичны шагу А1.)

А4. [Удаление нулевого члена.] Установить Q2 ← Q, LINK(Q1) ← Q ← LINK(Q) и AVAIL ⇐ Q2. (Порожденный на шаге А3 нулевой член удален из многочлена (Q).) Установить P ← LINK(P) и вернуться к шагу А2.

А5. [Вставка нового члена.] (Многочлен (P) содержит член, который отсутствует в многочлене (Q), поэтому его необходимо вставить в многочлен (Q).) Установить Q2 ⇐ AVAIL, COEF(Q2) ← COEF(P), ABC(Q2) ← ABC(P), LINK(Q2) ← Q, LINK(Q1) ← Q2, Q1 ← Q2, P ← LINK(P) и вернуться к шагу А2. ∎

Одна из наиболее замечательных особенностей алгоритма А заключается в способе следования указателя Q1 за указателем Q в списке. Этот способ типичен для алгоритмов обработки списков, и мы не раз еще встретим алгоритмы с такой же особенностью. Может ли читатель объяснить, почему данная идея использовалась в алгоритме А?

Читателю с небольшим опытом работы со связанными списками будет очень полезно внимательно изучить алгоритм А, например в качестве упражнения попробовать сложить многочлены $x + y + z$ и $x^2 - 2y - z$.

Зная алгоритм А, можно удивительно просто создать следующий алгоритм для операции умножения.

Алгоритм М (*Умножение многочленов*). Этот алгоритм аналогичен алгоритму А и заменяет многочлен (Q) суммой

$$\text{многочлен (Q)} + \text{многочлен (M)} \times \text{многочлен (P)}.$$

М1. [Следующий множитель.] Установить M ← LINK(M). Если ABC(M) < 0, то выполнение алгоритма прекращается.

М2. [Цикл умножения.] Выполнить алгоритм А, но всякий раз, когда появляется обозначение "ABC(P)", заменить его командой "если ABC(P) < 0, то -1;

в противном случае ABC(P)+ABC(M)"; когда появляется обозначение "COEF(P)", заменить его командой "COEF(P) × COEF(M)". Затем перейти к шагу M1. ▮

Программирование алгоритма A на языке компьютера MIX вновь демонстрирует легкость, с которой компьютер может манипулировать связанными списками. В приведенном ниже коде предполагается, что при возникновении переполнения (OVERFLOW) вызывается подпрограмма, которая либо завершает выполнение программы (из-за нехватки памяти), либо находит свободное пространство и выходит на rJ − 2.

Программа A (*Сложение многочленов*). Эта подпрограмма (рис. 10) создана так, чтобы ее можно было использовать вместе с подпрограммой умножения (см. упр. 15).

Вызов: JMP ADD.
Состояние до вызова: rI1 = P, rI2 = Q.
Состояние после вызова: Многочлен (Q) заменяется суммой: многочлен (Q) + многочлен (P); rI1 и rI2 остаются неизменными; все другие регистры имеют неопределенное содержимое.

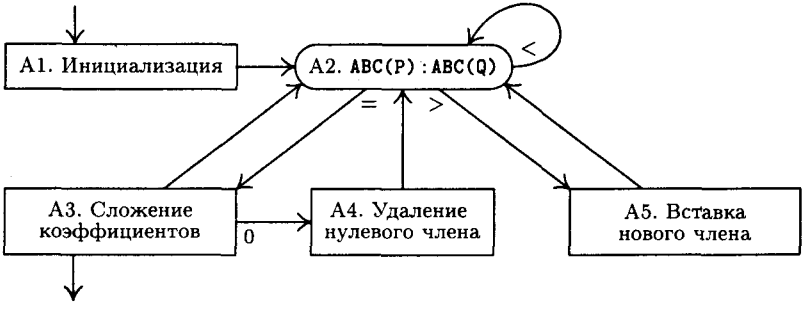


Рис. 10. Сложение многочленов.

В приведенном ниже коде P ≡ rI1, Q ≡ rI2, Q1 ≡ rI3 и Q2 ≡ rI6 для принятых в алгоритме A обозначений.

```
01  LINK EQU  4:5                Определение поля LINK.
02  ABC  EQU  0:3                Определение поля ABC.
03  ADD  STJ  3F          1      Вход в подпрограмму.
04  1H   ENT3 0,2         1+m''  A1. Инициализация. Установить Q1 ← Q.
05       LD2  1,3(LINK)   1+m''  Q ← LINK(Q1).
06  0H   LD1  1,1(LINK)   1+p    P ← LINK(P).
07  SW1  LDA  1,1         1+p    rA(0:3) ← ABC(P).
08  2H   CMPA 1,2(ABC)      x   A2. ABC(P):ABC(Q).
09       JE   3F            x   Если равно, перейти к шагу A3.
10       JG   5F          p'+q' Если больше, перейти к шагу A5.
11       ENT3 0,2           q'  Если меньше, установить Q1 ← Q.
12       LD2  1,3(LINK)     q'  Q ← LINK(Q1).
13       JMP  2B            q'  Повторить.
14  3H   JAN  *           m+1   A3. Сложение коэффициентов.
15  SW2  LDA  0,1             m  COEF(P)
16       ADD  0,2             m     + COEF(Q)
```

```
17          STA  0,2          m        → COEF(Q).
18          JANZ 1B           m        Если не равно нулю, совершить переход.
19          ENT6 0,2          m'       A4. Удаление нулевого члена. Q2 ← Q.
20          LD2  1,2(LINK)    m'       Q ← LINK(Q).
21          LDX  AVAIL        m'       ⎫
22          STX  1,6(LINK)    m'       ⎬ AVAIL ⇐ Q2.
23          ST6  AVAIL        m'       ⎭
24          ST2  1,3(LINK)    m'       LINK(Q1) ← Q.
25          JMP  0B           m'       Перейти с продвижением указателя P.
26    5H    LD6  AVAIL        p'       ⎫ A5. Вставка нового члена.
27          J6Z  OVERFLOW     p'       ⎪
28          LDX  1,6(LINK)    p'       ⎬ Q2 ⇐ AVAIL.
29          STX  AVAIL        p'       ⎭
30          STA  1,6          p'       ABC(Q2) ← ABC(P).
31    SW3   LDA  0,1          p'       rA ← COEF(P).
32          STA  0,6          p'       COEF(Q2) ← rA.
33          ST2  1,6(LINK)    p'       LINK(Q2) ← Q.
34          ST6  1,3(LINK)    p'       LINK(Q1) ← Q2.
35          ENT3 0,6          p'       Q1 ← Q2.
36          JMP  0B           p'       Перейти с продвижением указателя P.  ∎
```

Обратите внимание, что в алгоритме A предусмотрен только однократный проход каждого списка, причем для их обработки не пришлось выполнять несколько циклов. Используя закон Кирхгофа, без особого труда найдем, что количество команд и время выполнения зависят от следующих четырех величин:

m' — количество подобных членов, которые взаимно сокращаются;
m'' — количество подобных членов, которые нельзя сократить;
p' — количество членов в многочлене (P), для которых нет подобных членов в многочлене (Q);
q' — количество членов в многочлене (Q), для которых нет подобных членов в многочлене (P).

Анализ программы A с помощью обозначений

$$m = m' + m'', \quad p = m + p', \quad q = m + q', \quad x = 1 + m + p' + q'$$

позволяет получить следующую оценку времени ее выполнения на компьютере MIX: $(27m' + 18m'' + 27p' + 8q' + 13)u$. Минимальное количество узлов в пуле, которые необходимы для выполнения алгоритма, равно $2 + p + q$, а максимальное — равно $2 + p + q + p'$.

УПРАЖНЕНИЯ

1. [*21*] В начале этого раздела было предложено представлять пустой циклический список с помощью указателя PTR = Λ. Однако согласно идеологии создания циклических списков для обозначения пустого списка более последовательно было бы использовать значение PTR = LOC(PTR). Упрощает ли такое условие выполнение операций (a)–(c), которые описаны в начале этого раздела?

2. [*20*] Нарисуйте схемы состояний "до" и "после", которые демонстрируют результат выполнения операции конкатенации (3) при условии, что PTR_1 и PTR_2 не равны Λ.

▸ **3.** [*20*] Каким будет результат выполнения операции (3), если оба указателя $\texttt{PTR}_1$ и $\texttt{PTR}_2$ направлены на узлы *одного и того же* циклического списка?

4. [*20*] Сформулируйте операции вставки и удаления, которые в результате позволят использовать список (4) как *стек*.

▸ **5.** [*21*] Создайте алгоритм, который обращает циклический список, т. е. направления всех стрелок на схеме (1).

6. [*18*] Нарисуйте схему представления следующих многочленов в виде списков: (a) $xz - 3$; (b) 0.

7. [*10*] В чем заключается преимущество расположения членов многочлена в порядке убывания значений поля ABC?

▸ **8.** [*10*] В чем заключается преимущество следования указателя Q1 за указателем Q в алгоритме A?

▸ **9.** [*23*] Будет ли алгоритм A функционировать правильно, если P = Q (т. е. оба указателя указывают на один и тот же многочлен)? В каком случае алгоритм M будет работать правильно: если P = M, если P = Q или если M = Q?

▸ **10.** [*20*] При создании алгоритмов в данном разделе предполагалось, что в многочленах используются три переменные x, y и z, а их степени по отдельности никогда не превышают $b-1$ (где b — размер байта в компьютере MIX). Вместо этого предположим, что осуществляется сложение и умножение многочленов от одной переменной, x, степень которой может достигать значения $b^3 - 1$. Какие изменения следует внести в алгоритмы A и M?

11. [*24*] (Назначение этого упражнения и многих следующих заключается в создании пакета подпрограмм арифметики многочленов для совместной работы с программой A.) Поскольку алгоритмы A и M изменяют многочлен (Q), иногда полезно иметь подпрограмму, которая делала бы копию исходного многочлена. Напишите программу для компьютера MIX со следующими заданными спецификациями.

Вызов: JMP COPY.
Состояние до вызова: rI1 = P.
Состояние после вызова: rI2 указывает на вновь созданный многочлен, равный многочлену (P); rI1 остается неизменным; другие регистры не определены.

12. [*21*] Сравните время выполнения программы из упр. 11 и время выполнения алгоритма A, если многочлен(Q) = 0.

13. [*20*] Напишите подпрограмму для компьютера MIX со следующими спецификациями.

Вызов: JMP ERASE.
Состояние до вызова: rI1 = P.
Состояние после вызова: Многочлен (P) добавляется в список AVAIL; значения других регистров не определены.

[*Замечание.* Эта подпрограмма может быть использована вместе с подпрограммой из упр. 11 в последовательности "LD1 Q; JMP ERASE; LD1 P; JMP COPY; ST2 Q", чтобы получить "многочлен (Q) ← многочлен (P)".]

14. [*22*] Создайте подпрограмму для компьютера MIX со следующими спецификациями.

Вызов: JMP ZERO.
Состояние до вызова: Не обусловлено.
Состояние после вызова: rI2 указывает на вновь созданный многочлен, равный 0; значения других регистров не определены.

15. [*24*] Создайте подпрограмму для компьютера MIX для выполнения алгоритма M со следующими спецификациями.

Вызов: JMP MULT.
Состояние до вызова: rI1 = P, rI2 = Q, rI4 = M.
Состояние после вызова: Многочлен(Q) ← многочлен(Q) + многочлен(M) × многочлен(P); значения регистров rI1, rI2, rI4 остаются неизменными; значения других регистров не определены.

[*Замечание.* Используйте программу A как подпрограмму, изменив значения SW1, SW2 и SW3.]

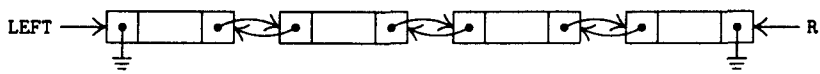
16. [*M28*] Оцените время выполнения подпрограммы из упр. 15 на основе наиболее важных параметров.

▶ **17.** [*22*] В чем заключается преимущество представления многочлена в виде циклического списка (описанного в этом разделе) по сравнению с представлением в виде прямого линейного связанного списка, который завершается Λ (и описанного в предыдущем разделе)?

▶ **18.** [*25*] Придумайте способ представления циклических списков внутри компьютера таким образом, чтобы проход списка был эффективным в обоих направлениях, причем чтобы в каждом узле использовалось только одно поле связи. [*Указание.* Если есть два указателя на два последовательных узла x_{i-1} и x_i, то должна существовать возможность доступа к узлам x_{i+1} и x_{i-2}.]

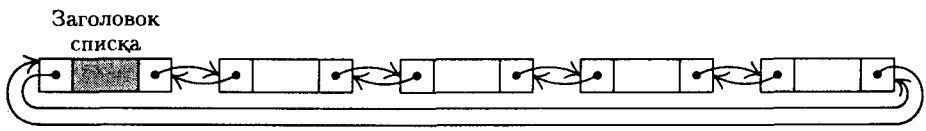
2.2.5. Дважды связанные списки

Для гибкой работы со связанными списками в каждый узел можно включить даже две связи, которые будут указывать на два соседних элемента:



(1)

Здесь LEFT и RIGHT обозначают переменные связи, которые указывают на левый и правый концы списка. Каждый узел списка включает две связи, например LLINK и RLINK. Как показано в упр. 1, при таком представлении со списком можно выполнять те же операции, что и с деком общего типа. Однако операции с деком почти всегда гораздо легче выполняются, если в нем присутствует узел с функциями *заголовка списка*, который рассмотрен в предыдущем разделе. При наличии заголовка списка типичная схема дважды связанного списка выглядит так, как показано ниже:



(2)

Поля RLINK и LLINK заголовка списка используются вместо указателей LEFT и RIGHT в схеме (1). Правый и левый концы абсолютно симметричны, поэтому заголовок списка может с таким же успехом располагаться с правой стороны списка на схеме (2). Если список пуст, оба поля связи заголовка списка указывают на сам заголовок.

Представление списка в виде (2), очевидно, удовлетворяет условию

$$\texttt{RLINK(LLINK(X))} = \texttt{LLINK(RLINK(X))} = \texttt{X}, \tag{3}$$

где X — адрес любого узла в списке (включая его заголовок). Именно поэтому представление списка в виде (2) предпочтительнее, чем в виде (1).

Дважды связанный список обычно занимает гораздо больше места в памяти, чем однократно связанный список (хотя в узле, который не заполняет полностью

слово в памяти компьютера, иногда остается свободное место для другой связи). Но дополнительные операции, которые могут очень эффективно выполняться с дважды связанными списками, часто являются более чем достаточной компенсацией за дополнительные затраты на память. Помимо очевидного преимущества, связанного с легкостью перехода либо назад, либо вперед вдоль дважды связанного списка, одной из наиболее принципиальных новинок является возможность *удаления узла* `NODE(X)` *из списка, если известно только значение* `X`. Эту операцию довольно легко вывести на основании схемы «состояния "до" и "после"» (рис. 11):

$$\begin{aligned} \texttt{RLINK(LLINK(X))} \leftarrow \texttt{RLINK(X)}, \qquad \texttt{LLINK(RLINK(X))} \leftarrow \texttt{LLINK(X)} \\ \texttt{AVAIL} \Leftarrow \texttt{X}. \end{aligned} \tag{4}$$

В списке с только односторонними ссылками нельзя удалить узел `NODE(X)`, не зная, какой узел ему предшествует, так как в предшествующем узле после удаления узла `NODE(X)` нужно соответствующим образом изменить значение его связи. В алгоритмах, рассмотренных в разделах 2.2.3 и 2.2.4, эта дополнительная информация всегда была под рукой при каждом удалении узла. В частности, в алгоритме 2.2.4A указатель `Q1` следовал за указателем `Q` именно для этой цели. Однако, как будет показано ниже, существуют алгоритмы, с помощью которых необходимо удалять произвольно выбранные узлы в середине списка. Именно для этого и используются дважды связанные списки. (Следует отметить, что из циклического списка можно удалить узел `NODE(X)`, зная только `X`, если выполнить полный замкнутый цикл переходов к предшественнику `X`. Ясно, что такая операция крайне неэффективна при работе с длинными списками, и потому она редко используется в качестве альтернативы для дважды связанного списка. См. ответ к упр. 2.2.4–8.)

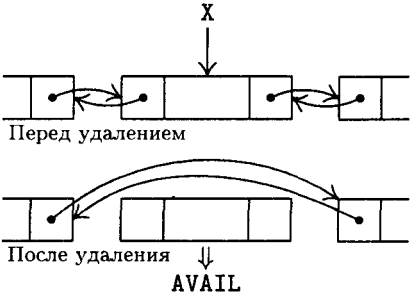


Рис. 11. Удаление узла из дважды связанного списка.

Так же просто в дважды связанный список можно вставить узел слева или справа от заданного узла `NODE(X)`. Например, для вставки узла справа от узла `NODE(X)` необходимо выполнить такие действия:

$$\begin{aligned} \texttt{P} \Leftarrow \texttt{AVAIL}, \quad \texttt{LLINK(P)} \leftarrow \texttt{X}, \quad \texttt{RLINK(P)} \leftarrow \texttt{RLINK(X)}, \\ \texttt{LLINK(RLINK(X))} \leftarrow \texttt{P}, \quad \texttt{RLINK(X)} \leftarrow \texttt{P}. \end{aligned} \tag{5}$$

Аналогично, меняя местами обозначения левых и правых элементов, получим соответствующий алгоритм для вставки нового элемента слева от заданного узла. Операция (5) изменяет значения пяти связей, поэтому она выполняется несколь-

ко медленнее, чем операция вставки в односвязном списке, где нужно изменить значения только для трех связей.

В качестве примера использования дважды связанных списков рассмотрим программу *дискретного моделирования*. "Дискретное моделирование" означает моделирование системы, в которой предполагается, что все изменения состояния системы происходят в некоторые дискретно заданные моменты Моделируемая "система" обычно представляет собой набор отдельных действующих лиц, которые, хотя и могут взаимодействовать друг с другом, в основном, ведут себя независимо. Например, это могут быть покупатели в магазине, корабли в гавани, сотрудники некоторого предприятия. При этом процесс моделирования заключается в выполнении определенных действий, предусмотренных для данного момента, для перехода к следующему моменту с дальнейшим выполнением других действий, запланированных для нового момента.

И наоборот, "непрерывное моделирование" означает моделирование действий, которые происходят непрерывно, например движение автомобилей по автостраде, полеты космических кораблей к другим планетам и т. д. Непрерывное моделирование часто можно вполне удовлетворительно имитировать с помощью дискретного моделирования с очень малыми временными интервалами между соседними шагами. Но в таком случае получится "синхронное" дискретное моделирование, при котором многие части системы слегка изменяются на каждом дискретном временном интервале, и такое приложение обычно нуждается в организации программы несколько иного типа, чем тот, который рассмотрен здесь.

Приведенная ниже программа моделирует работу лифта в здании факультета математики Калифорнийского технологического института (Калтех). Результаты такого моделирования, вероятно, будут полезны только тем, кому часто приходится посещать Калтех. И даже им, видимо, проще будет всего несколько раз воспользоваться этим лифтом, чем создавать специальную программу. Но, как обычно случается при изучении методов моделирования, используемые методы программирования гораздо интереснее, чем результаты выполнения программ. Рассматриваемые ниже методы иллюстрируют типичные методики, которые используются в программах дискретного моделирования.

Здание факультета математики имеет пять этажей: подвальный, цокольный, первый, второй и третий В нем находится один лифт с автоматическим управлением, который может останавливаться на каждом этаже. Для удобства перенумеруем этажи в следующем порядке: 0, 1, 2, 3 и 4

На каждом этаже есть две кнопки вызова лифта: одна — для движения вверх (`UP`), а другая — для движения вниз (`DOWN`). (На самом деле на этаже 0 имеется только кнопка `UP`, а на этаже 4 — только кнопка `DOWN`, но эти особые случаи будут игнорироваться, потому что дополнительные кнопки никогда не будут использоваться) Соответственно эти кнопки будут обозначаться десятью переменными `CALLUP[`j`]` и `CALLDOWN[`j`]`, $0 \leq j \leq 4$. Кроме того, переменные `CALLCAR[`j`]`, $0 \leq j \leq 4$, будут представлять кнопки внутри кабины лифта, которые обозначают этаж назначения. При нажатии кнопки соответствующей переменной присваивается значение 1, а после выполнения запроса (т е. после того как лифт достигнет заданного этажа) переменной присваивается значение 0.

До сих пор работа лифта описывалась с точки зрения пользователя, но ситуация станет более интересной, если рассмотреть ее с точки зрения лифта. Лифт может находиться в одном из трех следующих состояний: движение вниз (GOINGUP), движение вверх (GOINGDOWN) или нейтральное состояние (NEUTRAL) (Для человека текущее состояние обозначается светящимися стрелками внутри лифта.) Если лифт находится в нейтральном состоянии (NEUTRAL) и не на этаже 2, механизм лифта закроет двери и (если до закрытия дверей не дана никакая команда) лифт придет в состояние движения (GOINGUP или GOINGDOWN), направляясь к этажу 2. (Это его "базовый этаж", так как большинство людей входят в него именно здесь) Если лифт находится на этаже 2 в состоянии NEUTRAL, двери со временем закроются и лифт будет ожидать следующей команды. Первая полученная им команда для перемещения на другой этаж приведет лифт в состояние движения GOINGUP или GOINGDOWN в зависимости от нажатой кнопки. Лифт будет находиться в этом состоянии, пока не остановится. Тогда, если поступили новые команды, произойдет смена направления или переход в нейтральное состояние (NEUTRAL) непосредственно перед открытием дверей в зависимости от того, какие команды находятся в вызываемой последовательности. Лифту требуется некоторое время для открытия и закрытия дверей, ускорения и замедления, а также для перемещения от одного этажа к другому. Все эти величины указаны в приведенном ниже алгоритме, который выглядит гораздо более строго, чем привычные нам простые правила пользования лифтом. Этот алгоритм может и не отражать истинный принцип действия лифта, но автор все же верит, что он является простейшим набором правил, которые могут объяснить все те явления, которые автор наблюдал в ходе длительных экспериментов с лифтом во время написания этого раздела.

Работа лифта моделируется с помощью двух сопрограмм, одна из которых описывает поведение человека, а другая — лифта. Эти подпрограммы описывают все выполняемые действия, а также задержки времени, которые должны быть учтены при моделировании. В приведенном ниже описании переменная TIME представляет текущее значение моделируемых часов.

Все единицы времени даны в *десятых долях секунды*. Кроме того, в алгоритме используются другие переменные:

FLOOR, текущее положение кабины лифта;
D1, переменная, которая всегда равна нулю, за исключением промежутков времени, когда люди входят в лифт или выходят из него;
D2, переменная, которая становится равной нулю, если лифт более 30 с находится на одном из этажей без движения;
D3, переменная, которая всегда равна нулю, за исключением ситуации, когда двери открыты, но никто не входит в лифт и не выходит из него;
STATE, текущее состояние лифта (GOINGUP, GOINGDOWN или NEUTRAL).

В исходном состоянии FLOOR = 2, D1 = D2 = D3 = 0 и STATE = NEUTRAL.

Сопрограмма U (*Пользователи*) Каждый входящий в систему человек (т. е. тот, кто желает пользоваться лифтом) приступает к выполнению описанных ниже действий, начиная с шага U1.

U1. [Вход в систему, ожидание следующего человека.] Перечисленные ниже величины легко определить, но эти определения здесь не приводятся.

IN, этаж, на котором следующий человек входит в систему;
OUT, этаж, на который этот человек хочет попасть (OUT $\neq$ IN);
GIVEUPTIME, максимальное время ожидания человеком лифта, после которого его терпение исчерпывается и он решает воспользоваться лестницей;
INTERTIME, время до прихода другого человека.

После вычисления этих величин программа моделирования работы лифта организует весь процесс так, что следующий человек подходит к лифту в момент TIME + INTERTIME.

U2. [Получение сигнала и ожидание.] (Назначение этого шага заключается в вызове лифта; особые ситуации могут возникнуть, если лифт уже находится на нужном этаже.) Если FLOOR = IN и если следующее действие заключается в выполнении описанного ниже шага E6 (т. е. если двери лифта закрываются), нужно указать лифту на переход к шагу E3 и отменить выполнение шага E6. (Это значит, что двери откроются снова, прежде чем лифт начнет движение.) Если FLOOR = IN и D3 $\neq$ 0, установить D3 $\leftarrow$ 0, установить ненулевое значение для D1 и снова перейти к выполнению шага E4. (Это значит, что двери лифта откроются на том же этаже, но кто-то вошел или вышел. На шаге E4 людям предоставляется возможность входить в лифт в соответствии с обычными правилами хорошего тона. Следовательно, повторное возвращение к шагу E4 позволяет человеку войти в лифт до закрытия дверей.) Во всех других случаях человек устанавливает CALLUP[IN] $\leftarrow$ 1 или CALLDOWN[IN] $\leftarrow$ 1, а также OUT > IN или OUT < IN. Если же D2 = 0 или лифт находится в состоянии "спячки" E1, то выполняется указанная ниже подпрограмма DECISION. (Подпрограмма DECISION используется для вывода лифта из нейтрального состояния (NEUTRAL) в некоторые критические моменты.)

U3. [Вход в очередь.] Поместить человека в конец линейного списка QUEUE[IN], представляющего людей, которые ожидают лифта на том же этаже. Теперь человек должен терпеливо ожидать прихода лифта в течение времени GIVEUPTIME, а точнее — до тех пор, пока на шаге E4 программы будет совершен переход к шагу U5 и отменен запланированный шаг U4.

U4. [Отказ от длительного ожидания.] Если FLOOR $\neq$ IN или D1 = 0, удалить человека из списка QUEUE[IN] и из всей системы моделирования. (Человек решил, что лифт работает очень медленно или что небольшое физическое упражнение в виде ходьбы по лестнице все же лучше, чем езда на лифте.) Если FLOOR = IN и D1 $\neq$ 0, человек продолжает ждать (зная, что ожидание в этом случае не будет очень долгим во время выхода и входа других людей).

U5. [Вход в лифт.] Пользователь покидает список QUEUE[IN] и входит в лифт, т. е. список ELEVATOR со структурой стека, который представляет людей в кабине лифта. Установить CALLCAR[OUT] $\leftarrow$ 1.

Теперь, если STATE = NEUTRAL, установить значение STATE $\leftarrow$ GOINGUP или GOINGDOWN и перейти к шагу E5 через 25 единиц времени. (Эта особенность предусмотрена для того, чтобы двери могли закрыться быстрее, чем обычно, если в момент выбора человеком этажа назначения лифт находится в нейтральном состоянии. Такой временной интервал длиной 25 единиц предоставляет воз-

можность на шаге E4 убедиться в том, что значение переменной D1 правильно установлено к моменту выполнения шага E5, т. е. когда закрываются двери.)

Теперь человек ожидает момента перехода к выполнению шага U6 по окончании шага E4, когда лифт достигнет нужного этажа.

U6. [Выход из лифта.] Удалить пользователя из списка ELEVATOR и из системы моделирования. ▮

Сопрограмма E (*Лифт*). Эта сопрограмма представляет действия лифта; на шаге E4 также контролируется процесс входа людей в лифт и выхода из него.

E1. [Ожидание вызова.] (В этом случае лифт находится на этаже 2 с закрытыми дверями в состоянии ожидания.) Если кто-нибудь нажмет кнопку, подпрограмма DECISION совершит переход к шагу E3 или E6. В противном случае лифт находится в состоянии ожидания.

E2. [Изменение состояния?] Если STATE = GOINGUP и CALLUP[j] = CALLDOWN[j] = CALLCAR[j] = 0 для всех $j >$ FLOOR, то следует установить STATE $\leftarrow$ NEUTRAL или STATE $\leftarrow$ GOINGDOWN в зависимости от того, выполняется ли условие CALLCAR[j] = 0 для всех $j <$ FLOOR, и установить все переменные CALL для текущего этажа равными нулю. Если STATE = GOINGDOWN, то необходимо выполнить те же действия, но в обратном порядке.

E3. [Открытие дверей.] Задать для D1 и D2 любые ненулевые значения. По прошествии 300 единиц времени независимо выполнить шаг E9. (Эти действия могут быть отменены на шаге E6 до того, как они произойдут. Если они уже запланированы и не отменены, они отменяются и перепланируются.) Также независимо следует выполнить действия лифта на шаге E5 по прошествии 76 единиц времени. Затем следует подождать 20 единиц времени (для моделирования открытия дверей) и перейти к шагу E4.

E4. [Выход из лифта и вход в него людей.] Если для каких-то людей из списка ELEVATOR выполняется равенство OUT = FLOOR, то последнему зашедшему в лифт следует перейти к шагу U6, подождать на протяжении 25 единиц времени и повторить шаг E4. Если таких людей нет, но список QUEUE[FLOOR] не пуст, самому первому в списке следует немедленно перейти к шагу U5 вместо шага U4, подождать в течение 25 единиц и повторить шаг E4. А если список QUEUE[FLOOR] пуст, установить D1 $\leftarrow$ 0, установить D3 не равным нулю и подождать, пока какие-нибудь другие действия не вызовут продолжение выполнения алгоритма. (Действия на шаге E5 перенаправят нас к шагу E6 или действия на шаге U2 приведут к повторному переходу к шагу E4.)

E5. [Закрытие дверей.] Если D1 $\neq$ 0, подождать в течение 40 единиц и повторить данный шаг (двери при этом могут вздрогнуть, но вернуться в прежнее открытое положение, так как кто-то все еще входит или выходит). В противном случае установить D3 $\leftarrow$ 0 и перейти к шагу E6 через 20 единиц времени. (Так моделируется закрытие дверей после того, как все войдут в лифт или выйдут из него. Но если в ходе этого процесса на площадке перед лифтом на том же этаже появится новый человек, двери вновь откроются, как указано на шаге U2.)

E6. [Подготовка к движению.] Установить переменную CALLCAR[FLOOR] равной нулю; установить равной нулю переменную CALLUP[FLOOR], если STATE $\neq$

GOINGDOWN, а также установить равной нулю переменную CALLDOWN[FLOOR], если STATE $\neq$ GOINGUP. (*Замечание.* Если STATE = GOINGUP, лифт не сбрасывает значение CALLDOWN, так как предполагается, что люди, которым нужно ехать вниз, еще не вошли в него; см. упр. 6.) Теперь следует выполнить подпрограмму DECISION.

Если STATE = NEUTRAL даже после выполнения подпрограммы DECISION, то следует перейти к шагу E1. В противном случае, если D2 $\neq 0$, необходимо отменить действия лифта на шаге E9. Наконец, если STATE = GOINGUP, подождать в течение 15 единиц времени (чтобы лифт ускорился) и перейти к шагу E7; если STATE = GOINGDOWN, подождать в течение 15 единиц времени и перейти к шагу E8.

E7. [Подъем на этаж.] Установить FLOOR $\leftarrow$ FLOOR + 1 и подождать в течение 51 единицы времени. Если теперь CALLCAR[FLOOR] = 1 или CALLUP[FLOOR] = 1 или если ((FLOOR = 2 или CALLDOWN[FLOOR] = 1) и CALLUP[j] = CALLDOWN[j] = CALLCAR[j] = 0 для всех $j >$ FLOOR), подождать в течение 14 единиц времени (чтобы лифт притормозил) и перейти к шагу E2. В противном случае повторить данный шаг.

E8. [Спуск на этаж.] Этот шаг подобен шагу E7, если поменять направление движения, а временные интервалы длительностью 51 и 14 единиц заменить интервалами длительностью 61 и 23 единицы соответственно. (На спуск лифту требуется больше времени, чем на подъем.)

E9. [Установка индикатора бездействия.] Установить D2 $\leftarrow 0$ и выполнить подпрограмму DECISION. (Это независимое действие инициируется на шаге E3, но оно почти всегда отменяется на шаге E6. См. упр. 4.) ▮

Подпрограмма D (*Подпрограмма* DECISION). Как указано в приведенных выше сопрограммах, эта подпрограмма выполняется в критические моменты времени, когда должно быть принято решение о выборе направления движения.

D1. [Необходимо ли принять решение?] Если STATE $\neq$ NEUTRAL, выйти из этой подпрограммы.

D2. [Следует ли открыть двери лифта?] Если лифт находится на стадии выполнения действий шага E1 и если CALLUP[2], CALLCAR[2] и CALLDOWN[2] не равны нулю, то нужно после паузы длиной 20 единиц времени перейти к шагу E3, а затем выйти из этой подпрограммы. (Если в данный момент подпрограмма DECISION вызывается во время выполнения некоторых независимых действий на шаге E9, то сопрограмма лифта может перейти к шагу E1.)

D3. [Есть ли вызовы?] Найти наименьшее значение $j \neq$ FLOOR, для которого CALLUP[j], CALLCAR[j] или CALLDOWN[j] не равно нулю, и перейти к шагу D4. Но если такое значение j отсутствует, следует установить $j \leftarrow 2$, если в данный момент подпрограмма DECISION вызывается на шаге E6; в противном случае следует выйти из подпрограммы.

D4. [Изменение состояния.] Если FLOOR $> j$, установить STATE $\leftarrow$ GOINGDOWN; если FLOOR $< j$, установить STATE $\leftarrow$ GOINGUP.

D5. [Лифт ожидает?] Если сопрограмма лифта находится на шаге E1 и $j \neq 2$, то по прошествии 20 единиц времени перевести лифт к шагу E6. Выйти из подпрограммы. ▮

Эта модель работы лифта гораздо сложнее других описанных выше алгоритмов, но взятый из повседневной жизни пример более типичен для задач моделирования, чем любой состряпанный на скорую руку "учебный пример".

Для понимания принципа работы такого лифта рассмотрим табл. 1, которая представляет собой фрагмент одной из многих попыток моделирования. Вероятно, проще всего начать с проверки простого случая, например с момента 4257: лифт находится в состоянии покоя на этаже 2 с закрытыми дверями во время появления человека по имени Дон (в момент 4384). Две секунды спустя двери открываются и Дон входит в кабину лифта еще через две секунды. Нажав кнопку "3", он отправляет лифт вверх и наконец выходит на этаже 3, а лифт возвращается на этаж 2.

В первых строках табл. 1 показан более драматичный сценарий: человек вызывает лифт на цокольный этаж, но не дожидается его и уходит спустя 15,2 с. Лифт останавливается на цокольном этаже, но, так как там никого нет, направляется на этаж 4, поскольку несколько вызовов поступило со стороны людей, которые направляются вниз.

Программирование этой модели работы лифта для некоторого компьютера (в данном случае для компьютера MIX) заслуживает внимательного изучения. В любой момент в системе может существовать большое количество моделируемых людей (которые могут находиться в разных очередях, причем они могут не дождаться лифта и уйти в любое время). Кроме того, если сразу несколько человек пытается выйти из кабины лифта во время закрытия дверей, может получиться так, что действия, предусмотренные шагами E4, E5 и E9, будут выполняться одновременно. Течение времени и управление "одновременным" выполнением может быть запрограммировано в результате представления каждого элемента модели некоторым узлом, содержащим поле NEXTTIME (в нем указывается, когда выполняется следующее действие этого элемента) и поле NEXTINST (в нем хранится адрес, по которому данный элемент начинает выполнять команды аналогично ссылке для обычной сопрограммы). Каждый элемент ожидает наступления следующего момента в дважды связанном списке WAIT. Так как этот "список действий" сортируется по полю NEXTTIME, все действия будут обработаны в правильной последовательности моделируемых моментов. В программе также используются дважды связанные списки ELEVATOR и QUEUE.

Каждый узел, представляющий некоторое действие (т. е. действие человека или лифта), имеет следующий вид:

+	IN	LLINK1		RLINK1
+	NEXTTIME			
+	NEXTINST	0	0	39
+	OUT	LLINK2		RLINK2

(6)

Таблица 1

НЕКОТОРЫЕ ДЕЙСТВИЯ В МОДЕЛИ РАБОТЫ ЛИФТА

TIME	STATE	FLOOR	D1	D2	D3	Шаг	Действие
0000	N	2	0	0	0	U1	Человек 1 появляется на этаже 0 и хочет попасть на этаж 2.
0035	D	2	0	0	0	E8	Лифт движется вниз.
0038	D	1	0	0	0	U1	Человек 2 появляется на этаже 4 и хочет попасть на этаж 1.
0096	D	1	0	0	0	E8	Лифт движется вниз.
0136	D	0	0	0	0	U1	Человек 3 появляется на этаже 2 и хочет попасть на этаж 2.
0141	D	0	0	0	0	U1	Человек 4 появляется на этаже 2 и хочет попасть на этаж 1.
0152	D	0	0	0	0	U4	Человек 1 отказывается ждать лифт и уходит.
0180	D	0	0	0	0	E2	Лифт останавливается.
0180	N	0	0	X	0	E3	Двери лифта начинают открываться.
0200	N	0	X	X	0	E4	Двери открыты, но никого нет.
0256	N	0	0	X	X	E5	Двери лифта начинают закрываться.
0291	U	0	0	X	0	U1	Человек 5 появляется на этаже 3 и хочет попасть на этаж 1.
0291	U	0	0	X	0	E7	Лифт движется вверх.
0342	U	1	0	X	0	E7	Лифт движется вверх.
0364	U	2	0	X	0	U1	Человек 6 появляется на этаже 2 и хочет попасть на этаж 1.
0393	U	2	0	X	0	E7	Лифт движется вверх.
0444	U	3	0	X	0	E7	Лифт движется вверх.
0509	U	4	0	X	0	E2	Лифт останавливается.
0509	N	4	0	X	0	E3	Двери лифта начинают открываться.
0529	N	4	X	X	0	U5	Человек 2 заходит в кабину лифта.
0510	D	4	X	X	0	U4	Человек 6 отказывается ждать лифт и уходит.
0554	D	4	0	X	X	E5	Двери лифта начинают закрываться.
0589	D	4	0	X	0	E8	Лифт движется вниз.
0602	D	3	0	X	0	U1	Человек 7 появляется на этаже 1 и хочет попасть на этаж 2.
0673	D	3	0	X	0	E2	Лифт останавливается.
0673	D	3	0	X	0	E3	Двери лифта начинают открываться.
0693	D	3	X	X	0	U5	Человек 5 входит в кабину лифта.
0719	D	3	0	X	X	E5	Двери лифта начинают закрываться.
0784	D	3	0	X	0	E8	Лифт движется вниз.
0827	D	2	0	X	0	U1	Человек 8 появляется на этаже 1 и хочет попасть на этаж 0.
0868	D	2	0	X	0	E2	Лифт останавливается.
0868	D	2	0	X	0	E3	Двери лифта начинают открываться.
0876	D	2	X	X	0	U1	Человек 9 появляется на этаже 1 и хочет попасть на этаж 3.
0888	D	2	X	X	0	U5	Человек 3 входит в кабину лифта.
0913	D	2	X	X	0	U5	Человек 4 входит в кабину лифта.
0944	D	2	0	X	X	E5	Двери лифта начинают закрываться.
0979	D	2	0	X	0	E8	Лифт движется вниз.
1048	D	1	0	X	0	U1	Человек 10 появляется на этаже 0 и хочет попасть на этаж 4
1063	D	1	0	X	0	E2	Лифт останавливается.
1063	D	1	0	X	0	E3	Двери лифта начинают открываться.
1083	D	1	X	X	0	U6	Человек 4 выходит из кабины лифта и уходит.
1108	D	1	X	X	0	U6	Человек 3 выходит из кабины лифта и уходит.
1133	D	1	X	X	0	U6	Человек 5 выходит из кабины лифта и уходит.
1139	D	1	X	X	0	E5	Двери лифта вздрагивают.
1158	D	1	X	X	0	U6	Человек 2 выходит из кабины лифта и уходит.
1179	D	1	X	X	0	E5	Двери лифта вздрагивают.
1183	D	1	X	X	0	U5	Человек 7 входит в кабину лифта.
1208	D	1	X	X	0	U5	Человек 8 входит в кабину лифта.
1219	D	1	X	X	0	E5	Двери лифта вздрагивают.
1233	D	1	X	X	0	U5	Человек 9 входит в кабину лифта.
1259	D	1	0	X	X	E5	Двери лифта начинают закрываться.
1291	D	1	0	X	0	E8	Лифт движется вниз.
1378	D	0	0	X	0	E2	Лифт останавливается.
1378	U	0	0	X	0	E3	Двери лифта начинают открываться.
1398	U	0	X	X	0	U6	Человек 8 выходит из кабины лифта и уходит.
1423	U	0	X	X	0	U5	Человек 10 входит в кабину лифта.
1454	U	0	0	X	X	E5	Двери лифта начинают закрываться.
1489	U	0	0	X	0	E7	Лифт движется вверх.
1554	U	1	0	X	0	E2	Лифт останавливается.
1554	U	1	0	X	0	E3	Двери лифта начинают открываться.
1630	U	1	0	X	X	E5	Двери лифта начинают закрываться.
1665	U	1	0	X	0	E7	Лифт движется вверх.
..							
4257	N	2	0	X	0	E1	Лифт находится в состоянии ожидания.
4384	N	2	0	X	0	U1	Человек 17 появляется на этаже 2 и хочет попасть на этаж 3.
4404	N	2	0	X	0	E3	Двери лифта начинают открываться.
4424	N	2	X	X	0	U5	Человек 17 входит в кабину лифта.
4449	U	2	0	X	X	E5	Двери лифта начинают закрываться.
4484	U	2	0	X	0	E7	Лифт движется вверх.
4549	U	3	0	X	0	E2	Лифт останавливается.
4549	N	3	0	X	0	E3	Двери лифта начинают открываться.
4569	N	3	X	X	0	U6	Человек 17 выходит из кабины лифта и уходит.
4625	N	3	0	X	X	E5	Двери лифта начинают закрываться.
4660	D	3	0	X	0	E8	Лифт движется вниз.
4744	D	2	0	X	0	E2	Лифт останавливается
4744	N	2	0	X	0	E3	Двери лифта начинают открываться.
4764	N	2	X	X	0	E4	Двери лифта открыты, на площадке никого нет.
4820	N	2	0	X	0	E5	Двери лифта начинают закрываться.
4840	N	2	0	X	0	E1	Лифт находится в состоянии ожидания.
...							

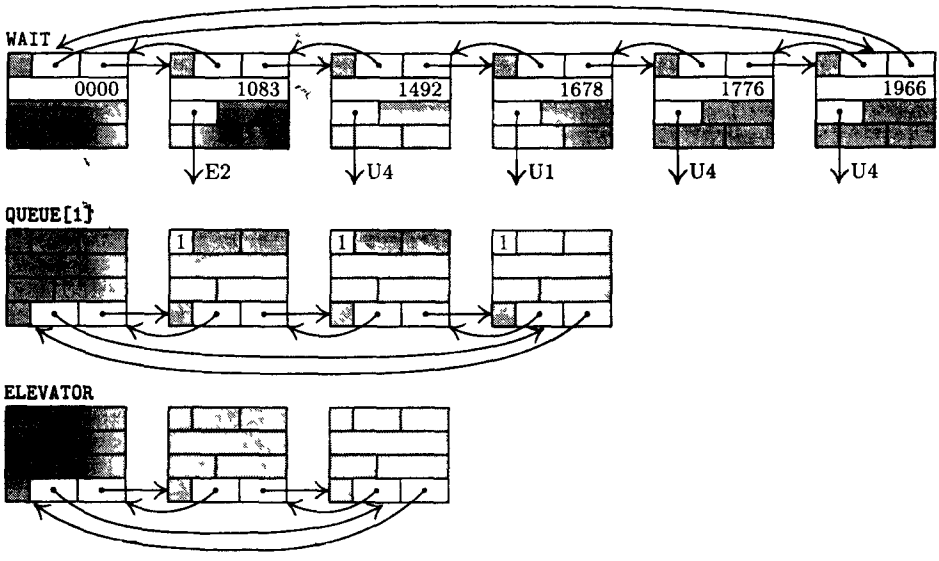


Рис. 12. Некоторые списки из программы моделирования работы лифта (Заголовки списков указаны слева)

Здесь LLINK1 и RLINK1 — связи в списке WAIT, а LLINK2 и RLINK2 — связи в списках QUEUE и ELEVATOR Последние два поля, а также поля IN и OUT в структуре (6) относятся к узлу, который представляет человека, и не имеют никакого отношения к узлу, представляющему лифт. Третье слово этого узла на самом деле является командой "JMP" на языке компьютера MIX

На рис. 12 показано типичное содержимое списков WAIT, ELEVATOR и одного из списков QUEUE Каждый узел в списке QUEUE одновременно находится в списке WAIT со значением NEXTINST = U4, но это не отражено на данном рисунке, поскольку сложность всех взаимосвязей может помешать читателю понять основную идею.

Теперь рассмотрим саму программу. Она довольна большая, но делится на несколько малых частей (как это обычно бывает со всеми большими программами), каждая из которых выглядит достаточно просто Сначала идет некоторое количество строк кода с определением начального содержания таблиц. Рассмотрим наиболее интересные из них, а именно — заголовки списка WAIT (строки 010 и 011), списка QUEUE (строки 026 и 031) и списка ELEVATOR (строки 032–033). Каждый из них является узлом со структурой (6), в которой удалены ненужные слова Заголовок списка WAIT содержит только два первых слова узла, а заголовкам списков QUEUE и ELEVATOR требуется только последнее слово узла. Кроме того, имеются четыре узла, которые всегда присутствуют в данной модели (строки 012–023): узел USER1, который всегда представляет шаг U1, подготовку к появлению в системе нового человека; узел ELEV1, который управляет основными действиями лифта на шагах E1–E4, E6–E8, а также узлы ELEV2 и ELEV3, которые используются для действий лифта на шагах E5 и E9, которые выполняются независимо от других действий лифта относительно моделируемой шкалы времени. Каждый из этих четырех узлов содержит только три слова, так как они никогда не будут входить в состав списков

QUEUE и ELEVATOR. Узлы, представляющие деятельность людей, будут находиться в пуле после основной программы.

```
001  * МОДЕЛЬ РАБОТЫ ЛИФТА
002  IN        EQU  1:1                    Определение полей
003  LLINK1    EQU  2:3                        внутри узлов.
004  RLINK1    EQU  4:5
005  NEXTINST  EQU  0:2
006  OUT       EQU  1:1
007  LLINK2    EQU  2:3
008  RLINK2    EQU  4:5

009  * ТАБЛИЦЫ ФИКСИРОВАННОГО РАЗМЕРА И ЗАГОЛОВКИ СПИСКОВ
010  WAIT      CON  *+2(LLINK1),*+2(RLINK1)  Заголовок списка WAIT,
011            CON  0                            где всегда NEXTTIME = 0.
012  USER1     CON  *-2(LLINK1),*-2(RLINK1)  Это узел действия U1,
013            CON  0                            сначала только он
014            JMP  U1                           находится в списке WAIT.
015  ELEV1     CON  0                        Этот узел представляет
016            CON  0                            действия лифта, кроме
017            JMP  E1                           действий на шагах E5 и E9.
018  ELEV2     CON  0                        Этот узел представляет
019            CON  0                            независимые действия
020            JMP  E5                           лифта на шаге E5.
021  ELEV3     CON  0                        Этот узел представляет
022            CON  0                            независимые действия
023            JMP  E9                           лифта на шаге E9.
024  AVAIL     CON  0                        Связь со свободными узлами.
025  TIME      CON  0                        Текущее время моделирования.
026  QUEUE     EQU  *-3
027            CON  *-3(LLINK2),*-3(RLINK2)  Заголовок списка QUEUE[0].
028            CON  *-3(LLINK2),*-3(RLINK2)  Заголовок списка QUEUE[1].
029            CON  *-3(LLINK2),*-3(RLINK2)  Сначала все очереди
030            CON  *-3(LLINK2),*-3(RLINK2)      пусты.
031            CON  *-3(LLINK2),*-3(RLINK2)  Заголовок списка QUEUE[4].
032  ELEVATOR  EQU  *-3
033            CON  *-3(LLINK2),*-3(RLINK2)  Заголовок списка ELEVATOR.
034            CON  0                        }
035            CON  0                        } "Дополнение" нулями таблицы CALL
036            CON  0                        }     (см. строки 183–186).
037            CON  0                        }
038  CALL      CON  0                    CALLUP[0], CALLCAR[0], CALLDOWN[0]
039            CON  0                    CALLUP[1], CALLCAR[1], CALLDOWN[1]
040            CON  0                    CALLUP[2], CALLCAR[2], CALLDOWN[2]
041            CON  0                    CALLUP[3], CALLCAR[3], CALLDOWN[3]
042            CON  0                    CALLUP[4], CALLCAR[4], CALLDOWN[4]
043            CON  0                        }
044            CON  0                        } "Дополнение" нулями таблицы CALL
045            CON  0                        }     (см  строки 178–181).
046            CON  0                        }
047  D1        CON  0                    Индикация открытых дверей,
                                               активное состояние.
```

```
048  D2      CON  0          Индикация состояния бездействия.
049  D3      CON  0          Индикация открытых дверей,
                                 активное состояние ▌
```

Следующая часть программы содержит основные подпрограммы и главные управляющие процедуры процесса моделирования. Подпрограммы INSERT и DELETE выполняют типичные манипуляции дважды связанными списками. Они помещают текущий узел в список QUEUE или ELEVATOR либо извлекают его из этих списков. (В программе "текущий узел" C всегда представлен индексным регистром 6.) Для работы со списком WAIT также предусмотрены некоторые подпрограммы. Так, подпрограмма SORTIN помещает текущий узел в список WAIT, сортируя его по полю NEXTTIME. А подпрограмма IMMED вставляет текущий узел в середину списка WAIT. Подпрограмма HOLD помещает текущий узел в список WAIT, причем значение поля NEXTTIME равно текущему времени плюс значение регистра A. Подпрограмма DELETEW удаляет текущий узел из списка WAIT.

Процедура CYCLE является сердцевиной модели управления, поскольку именно она определяет, какое действие будет выполнено дальше (а именно — первый элемент списка WAIT, который, как нам теперь известно, не пуст), и переходит к его выполнению. Существует два особых входа в процедуру CYCLE: во-первых, CYCLE1 устанавливает значение NEXTINST в текущем узле, а во-вторых, HOLDC выполняет то же самое с дополнительным вызовом подпрограммы HOLD. Таким образом, результат выполнения команды "JMP HOLDC" со значением t в регистре A будет заключаться в приостановке действий на t единиц по моделируемой шкале времени с переходом в дальнейшем к следующему адресу.

```
050  * ПОДПРОГРАММЫ И ПРОЦЕДУРА УПРАВЛЕНИЯ
051  INSERT  STJ  9F             Вставить NODE(C) слева от NODE(rI1).
052          LD2  3,1(LLINK2)    rI2 ← LLINK2(rI1).
053          ST2  3,6(LLINK2)    LLINK2(C) ← rI2.
054          ST6  3,1(LLINK2)    LLINK2(rI1) ← C.
055          ST6  3,2(RLINK2)    RLINK2(rI2) ← C.
056          ST1  3,6(RLINK2)    RLINK2(C) ← rI1.
057  9H      JMP  *              Выйти из подпрограммы.
058  DELETE  STJ  9F             Удалить NODE(C) из списка.
059          LD1  3,6(LLINK2)    P ← LLINK2(C)
060          LD2  3,6(RLINK2)    Q ← RLINK2(C).
061          ST1  3,2(LLINK2)    LLINK2(Q) ← P.
062          ST2  3,1(RLINK2)    RLINK2(P) ← Q
063  9H      JMP  *              Выйти из подпрограммы.
064  IMMED   STJ  9F             Вставить NODE(C) в начале списка WAIT.
065          LDA  TIME
066          STA  1,6            Установить NEXTTIME(C) ← TIME.
067          ENT1 WAIT           P ← LOC(WAIT)
068          JMP  2F             Вставить NODE(C) справа от NODE(P).
069  HOLD    ADD  TIME           rA ← TIME + rA.
070  SORTIN  STJ  9F             Рассортировать NODE(C) в списке WAIT.
071          STA  1,6            Установить NEXTTIME(C) ← rA.
072          ENT1 WAIT           P ← LOC(WAIT)
073          LD1  0,1(LLINK1)    P ← LLINK1(P)
074          CMPA 1,1            Сравнить значения в полях NEXTTIME слева направо.
```

```
075             JL    *-2               Повторять до тех пор,
                                            пока NEXTTIME(C) ≥ NEXTTIME(P).
076  2H         LD2   0,1(RLINK1)       Q ← RLINK1(P).
077             ST2   0,6(RLINK1)       RLINK1(C) ← Q.
078             ST1   0,6(LLINK1)       LLINK1(C) ← P.
079             ST6   0,1(RLINK1)       RLINK1(P) ← C.
080             ST6   0,2(LLINK1)       LLINK1(Q) ← C
081  9H         JMP   *                 Выйти из подпрограммы.
082  DELETEW    STJ   9F                Удалить NODE(C) из списка WAIT.
083             LD1   0,6(LLINK1)       (Такой же код, как в строках 058–063,
084             LD2   0,6(RLINK1)           но LLINK1 и RLINK1 используются
085             ST1   0,2(LLINK1)           вместо LLINK2 и RLINK2.)
086             ST2   0,1(RLINK1)
087  9H         JMP   *
088  CYCLE1     STJ   2,6(NEXTINST)     Установить NEXTINST(C) ← rJ.
089             JMP   CYCLE
090  HOLDC      STJ   2,6(NEXTINST)     Установить NEXTINST(C) ← rJ.
091             JMP   HOLD              Вставить NODE(C) в список WAIT с задержкой rA.
092  CYCLE      LD6   WAIT(RLINK1)      Установить текущий узел C ← RLINK1(LOC(WAIT)).
093             LDA   1,6
094             STA   TIME              TIME ← NEXTTIME(C).
095             JMP   DELETEW           Удалить NODE(C) из списка WAIT.
096             JMP   2,6               Перейти к NEXTINST(C). ▮
```

Теперь рассмотрим код сопрограммы U. В начале шага U1 текущим узлом C является узел USER1 (см. выше строки 012–014) и указанные в строках 099 и 100 действия повторно помещают человека USER1 в список WAIT так, что следующий человек появится в системе спустя INTERTIME единиц времени. В строках 101–114 создается узел для этого нового человека и записываются этажи входа (IN) и выхода (OUT). Стек AVAIL является односвязным по полю RLINK1 каждого узла. Обратите внимание, что в строках 101–108 действие "C ⇐ AVAIL" выполняется с помощью метода POOLMAX, 2.2.3–(7); при этом вряд ли здесь понадобится проверка события переполнения (OVERFLOW), так как общий размер пула (количество людей в данной модели работы лифта в произвольный момент времени) едва ли превышает 10 узлов (40 слов). Возврат узла в стек AVAIL представлен в строках 156–158.

Во всей этой программе индексный регистр 4 равен переменной FLOOR, а индексный регистр 5 положителен, отрицателен или равен нулю в зависимости от текущего состояния, т. е. при выполнении условия STATE = GOINGUP, STATE = GOINGDOWN или STATE = NEUTRAL соответственно. Переменные CALLUP[j], CALLCAR[j] и CALLDOWN[j] занимают соответственно поля (1:1), (3:3) и (5:5) ячеек CALL + j.

```
097  * СОПРОГРАММА U                  U1. Вход в систему, ожидание следующего человека.
098  U1   JMP   VALUES                Вычислить IN, OUT, GIVEUPTIME, INTERTIME
099       LDA   INTERTIME              INTERTIME вычисляется подпрограммой VALUES
100       JMP   HOLD                   Разместить NODE(C) в списке WAIT
                                           с задержкой INTERTIME.
101       LD6   AVAIL                  C ← AVAIL.
102       J6P   1F                     Если AVAIL ≠ Λ, совершить переход
103       LD6   POOLMAX(0:2)
104       INC6  4                      C ← POOLMAX + 4.
```

```
105         ST6  POOLMAX(0:2)      POOLMAX ← C.
106         JMP  *+3               Предполагается, что переполнение не происходит.
107   1H    LDA  0,6(RLINK1)
108         STA  AVAIL             AVAIL ← RLINK1(AVAIL).
109         LD1  INFLOOR           rI1 ← INFLOOR (вычисляется подпрограммой VALUES).
110         ST1  0,6(IN)           IN(C) ← rI1.
111         LD2  OUTFLOOR          rI2 ← OUTFLOOR
                                       (вычисляется подпрограммой VALUES).
112         ST2  3,6(OUT)          OUT(C) ← rI2.
113         ENTA 39                Поместить константу 39 (код операции JMP)
114         STA  2,6                   в третье слово структуры (6).
115   U2    ENTA 0,4               U2. Получение сигнала и ожидание.
                                       Установить rA ← FLOOR.
116         DECA 0,1               FLOOR − IN.
117         ST6  TEMP              Сохранить значение C.
118         JANZ 2F                Совершить переход, если FLOOR ≠ IN.
119         ENT6 ELEV1             Установить C ← LOC(ELEV1).
120         LDA  2,6(NEXTINST)     Находится ли лифт на шаге E6?
121         DECA E6
122         JANZ 3F
123         ENTA E3                Если да, перейти к шагу E3.
124         STA  2,6(NEXTINST)
125         JMP  DELETEW           Удалить его из списка WAIT
126         JMP  4F                    и повторно вставить в начало списка WAIT.
127   3H    LDA  D3
128         JAZ  2F                Совершить переход, если D3 = 0.
129         ST6  D1                В противном случае присвоить D1
                                       ненулевое значение.
130         STZ  D3                Установить D3 ← 0.
131   4H    JMP  IMMED             Вставить ELEV1 в начало списка WAIT.
132         JMP  U3                (rI1 и rI2 изменены.)
133   2H    DEC2 0,1               rI2 ← OUT − IN.
134         ENTA 1
135         J2P  *+3               Совершить переход, если лифт направляется вверх.
136         STA  CALL,1(5:5)       Установить CALLDOWN[IN] ← 1.
137         JMP  *+2
138         STA  CALL,1(1:1)       Установить CALLUP[IN] ← 1.
139         LDA  D2
140         JAZ  *+3               Если D2 = 0, вызвать подпрограмму DECISION.
141         LDA  ELEV1+2(NEXTINST)
142         DECA E1                Если лифт находится на шаге E1, вызвать
143         JAZ  DECISION              подпрограмму DECISION.
144   U3    LD6  TEMP              U3. Вход в очередь.
145         LD1  0,6(IN)
146         ENT1 QUEUE,1           rI1 ← LOC(QUEUE[IN]).
147         JMP  INSERT            Вставить NODE(C) с правого конца QUEUE[IN].
148   U4A   LDA  GIVEUPTIME
149         JMP  HOLDC             Подождать GIVEUPTIME единиц времени.
150   U4    LDA  0,6(IN)           U4. Отказ от длительного ожидания.
151         DECA 0,4               IN(C) − FLOOR.
152         JANZ *+3
```

```
153         LDA  D1              FLOOR = IN(C).
154         JANZ U4A             См. упр. 7.
155  U6     JMP  DELETE          U6. Выход из лифта. NODE(C) удаляется
156         LDA  AVAIL               из QUEUE или ELEVATOR.
157         STA  0,6(RLINK1)     AVAIL ⇐ C.
158         ST6  AVAIL
159         JMP  CYCLE           Продолжить моделирование.
160  U5     JMP  DELETE          U5. Вход в лифт. NODE(C) удаляется
161         ENT1 ELEVATOR            из QUEUE.
162         JMP  INSERT          Вставить его с правой стороны списка ELEVATOR.
163         ENTA 1
164         LD2  3,6(OUT)
165         STA  CALL,2(3:3)     Установить CALLCAR[OUT(C)] ← 1.
166         J5NZ CYCLE           Совершить переход, если STATE ≠ NEUTRAL
167         DEC2 0,4             rI2 ← OUT(C) − FLOOR.
168         ENT5 0,2             Установить направление движения для STATE
169         ENT6 ELEV2           Установить C ← LOC(ELEV2).
170         JMP  DELETEW         Удалить шаг E5 из списка WAIT
171         ENTA 25
172         JMP  E5A             Повторить шаг E5 спустя 25 единиц времени. ▮
```

Код сопрограммы E представляет собой очень прямолинейную реализацию приведенного выше полуформального описания. Вероятно, наиболее интересная его часть связана с подготовкой независимых действий лифта на шаге E3 и поиском в списках ELEVATOR и QUEUE на шаге E4.

```
173  * СОПРОГРАММА E
174  E1A JMP  CYCLE1          Установить NEXTINST ← E1, перейти к CYCLE.
175  E1  EQU  *               E1. Ожидание вызова. (Никаких действий.)
176  E2A JMP  HOLDC
177  E2  J5N  1F              E2. Изменение состояния?
178      LDA  CALL+1,4        Движение вверх (GOINGUP).
179      ADD  CALL+2,4
180      ADD  CALL+3,4
181      ADD  CALL+4,4
182      JAP  E3              Есть ли вызовы на верхние этажи?
183      LDA  CALL-1,4(3:3)   Если нет, посылали ли пассажиры лифта
184      ADD  CALL-2,4(3:3)       запрос для перехода к нижним этажам?
185      ADD  CALL-3,4(3:3)
186      ADD  CALL-4,4(3:3)
187      JMP  2F
188  1H  LDA  CALL-1,4        Движение вниз GOINGDOWN.
189      ADD  CALL-2,4        Действия те же, что и в строках 178–186.
  ⋮
196      ADD  CALL+4,4(3:3)
197  2H  ENN5 0,5             Изменить направление движения в STATE.
198      STZ  CALL,4          Установить переменные CALL равными нулю.
199      JANZ E3              Совершить переход при вызове обратного
                                  направления;
200      ENT5 0               в противном случае установить STATE ← NEUTRAL.
```

```
201  E3   ENT6 ELEV3
202       LDA  0,6
203       JANZ DELETEW
204       ENTA 300
205       JMP  HOLD
206       ENT6 ELEV2
207       ENTA 76
208       JMP  HOLD
209       ST6  D2
210       ST6  D1
211       ENTA 20
212  E4A  ENT6 ELEV1
213       JMP  HOLDC
214  E4   ENTA 0,4
215       SLA  4
216       ENT6 ELEVATOR
217  1H   LD6  3,6(LLINK2)
218       CMP6 =ELEVATOR=
219       JE   1F
220       CMPA 3,6(OUT)
221       JNE  1B
222       ENTA U6
223       JMP  2F
224  1H   LD6  QUEUE+3,4(RLINK2)
225       CMP6 3,6(RLINK2)
226       JE   1F
227       JMP  DELETEW
228       ENTA U5
229  2H   STA  2,6(NEXTINST)
230       JMP  IMMED
231       ENTA 25
232       JMP  E4A
233  1H   STZ  D1
234       ST6  D3
235       JMP  CYCLE
236  E5A  JMP  HOLDC
237  E5   LDA  D1
238       JAZ  *+3
239       ENTA 40
240       JMP  E5A
241       STZ  D3
242       ENT6 ELEV1
243       ENTA 20
244       JMP  HOLDC
245  E6   J5N  *+2
246       STZ  CALL,4(1:3)
247       J5P  *+2
248       STZ  CALL,4(3:5)
```

E3. Открытие дверей.
Если шаг E9 уже запланирован,
удалить его из списка WAIT.

Запланировать шаг E9 спустя 300 единиц времени.

Запланировать шаг E5 спустя 76 единиц времени.
Установить D2 не равной нулю.
Установить D1 не равной нулю.

E4. Выход из лифта и вход в него.
Ввести значение FLOOR в поле OUT регистра rA.
C ← LOC(ELEVATOR).
C ← LLINK2(C).
Поиск в списке ELEVATOR справа налево.
Если C = LOC(ELEVATOR), завершить поиск.
Сравнить OUT(C) с FLOOR.
Если они не равны, продолжить поиск;
в противном случае приготовиться
к переходу человека к шагу U6.
Установить C ← RLINK2(LOC(QUEUE[FLOOR])).
Верно ли, что C = RLINK2(C)?
Если верно, то очередь пуста.
Если неверно, отменить шаг U4 для этого человека.
Подготовиться к замене шага U4 шагом U5.
Установить NEXTINST(C).
Поместить человека в начало списка WAIT.

Подождать 25 единиц времени и повторить шаг E4
Установить D1 ← 0.
Установить D3 не равной нулю.
Вернуться к моделированию других событий.

E5. Закрытие дверей.
Верно ли, что D1 = 0?
Если неверно, значит, люди все еще входят
или выходят.
Подождать 40 единиц времени и повторить шаг E5.
Если D1 = 0, установить D3 ← 0.

Подождать 20 единиц времени,
затем перейти к шагу E6.
E6. Подготовка к движению.
Если STATE ≠ GOINGDOWN,
то сбросить значения CALLUP и CALLCAR
для этого этажа.
Если ≠ GOINGUP, сбросить CALLCAR и CALLDOWN

```
249         J5Z   DECISION          Выполнить подпрограмму DECISION.
250   E6B   J5Z   E1A               Если STATE = NEUTRAL, перейти к шагу E1
251         LDA   D2                    и подождать.
252         JAZ   *+4
253         ENT6  ELEV3             В противном случае, если D2 ≠ 0,
254         JMP   DELETEW               отменить шаг E9
255         STZ   ELEV3                 (см. строку 202).
256         ENT6  ELEV1
257         ENTA  15                Подождать 15 единиц времени.
258         J5N   E8A               Если STATE = GOINGDOWN, перейти к шагу E8.
259   E7A   JMP   HOLDC
260   E7    INC4  1                 E7. Подъем на этаж.
261         ENTA  51
262         JMP   HOLDC             Подождать в течение 51 единицы времени.
263         LDA   CALL,4(1:3)       Верно ли, что CALLCAR[FLOOR]
264         JAP   1F                    или CALLUP[FLOOR] ≠ 0?
265         ENT1  -2,4              Если неверно,
266         J1Z   2F                    то верно ли, что FLOOR = 2?
267         LDA   CALL,4(5:5)       Если неверно, то верно ли, что CALLDOWN[FLOOR] ≠ 0?
268         JAZ   E7                Если неверно, повторить шаг E7.
269   2H    LDA   CALL+1,4
270         ADD   CALL+2,4
271         ADD   CALL+3,4
272         ADD   CALL+4,4
273         JANZ  E7                Есть ли вызовы на верхние этажи?
274   1H    ENTA  14                Пора остановить лифт.
275         JMP   E2A               Подождать в течение 14 единиц времени
276   E8A   JMP   HOLDC                и перейти к шагу E2.

  .                                 (См. упр. 8.)
  .
  .
292         JMP   E2A
293   E9    STZ   0,6               E9. Установка индикатора бездействия.
                                        (См. строку 202.)
294         STZ   D2                D2 ← 0.
295         JMP   DECISION          Выполнить подпрограмму DECISION.
296         JMP   CYCLE             Вернуться к моделированию других событий. ▮
```

Подпрограмма DECISION здесь не рассматривается (см. упр. 9), как и подпрограмма VALUES, которая применяется для указания запросов лифта. В самом конце программы приводится следующий код:

```
BEGIN    ENT4  2          В начале FLOOR = 2
         ENT5  0              и STATE = NEUTRAL.
         JMP   CYCLE      Начать моделирование.
POOLMAX  NOP   POOL
POOL     END   BEGIN      Вслед за пулом располагаются литералы,
                              временное хранилище. ▮
```

Приведенная выше программа прекрасно моделирует работу лифта, но выполнять ее бесполезно, поскольку она не выводит никаких выходных данных! На самом деле автор добавил в нее подпрограмму PRINT, которая вызывалась в наиболее

критические моменты; именно с ее помощью была подготовлена табл. 1. Эти подробности здесь опущены; они довольно просты, но в значительной мере усложняют код.

Было создано несколько языков программирования, которые позволяли упростить форму указания действий дискретного моделирования с последующей компиляцией этих указаний на машинный язык. В данном разделе язык ассемблера использовался, конечно же, потому, что здесь рассматривались основные методы управления связанными списками и подробности дискретного моделирования на компьютере, который способен выполнять последовательные, а не параллельные вычисления. Метод управления последовательностью сопрограмм на основе списка WAIT или перечня действий, который описан в этом разделе, называется *квазипараллельной обработкой* (*quasi-parallel processing*).

Анализ времени выполнения такой длинной программы сделать довольно трудно, потому что она пронизана весьма сложными взаимосвязями. Однако большие программы часто тратят бóльшую часть времени на выполнение сравнительно небольших процедур, которые осуществляют простые действия. Поэтому хорошую оценку общей производительности можно получить с помощью особой процедуры трассировки, или, иначе говоря, с помощью *профайлера* (*profiler*), которая выполняет программу и записывает, как часто выполняется та или иная команда. Она позволяет обнаружить "уязвимые места" производительности программы, которым следует уделить особое внимание. [См. упр. 1.4.3.2–7. См. также *Software Practice & Experience* **1** (1971), 105–133, где приводятся примеры таких исследований для программ на языке FORTRAN, произвольно выбранных из мусорных корзин Станфордского компьютерного центра (Stanford Computer Center).] Автор экспериментировал с программой моделирования работы лифта, запуская ее в течение 10000 единиц по моделируемой шкале времени и при условии, что в модели находится 26 человек. Оказалось, что наиболее часто выполнялся цикл сортировки SORTIN, строки 073–075, а именно — 1432 раза, тогда как процедура сортировки SORTIN вызывалась 437 раз. Процедура CYCLE выполнялась 407 раз, поэтому для увеличения скорости ее выполнения не следовало бы вызывать подпрограмму DELETEW в строке 095; лучше полностью включить в программу четыре строки этой подпрограммы (с экономией $4u$ при каждом использовании CYCLE). С помощью профайлера также было обнаружено, что подпрограмма DECISION вызывалась только 32 раза, а цикл на шаге E4 (строки 217–219) выполнялся всего 142 раза.

Автор надеется, что при изучении этого примера читатель узнает столько же нового о моделировании, сколько автор узнал о работе лифтов.

УПРАЖНЕНИЯ

1. [*21*] Предложите описание операций вставки и удаления данных с левого конца дважды связанного списка (1). (С учетом тех же операций на правом конце, которые можно вывести из соображений симметрии, получим описание всех действий для дека общего типа)

▸ **2.** [*22*] Объясните, почему для односвязного списка нельзя выполнять операции так же эффективно, как для дека общего типа Удаление элементов может быть эффективно выполнено только с одного конца односвязного списка

▸ **3.** [*22*] В модели работы лифта из данного раздела для каждого этажа используются три переменные вызова (CALLUP, CALLCAR и CALLDOWN), которые представляют нажимаемые кнопки. Вполне возможно, что вместо трех лифту необходима только одна или

две двоичные переменные для кнопок вызова на каждом этаже. Объясните, в какой последовательности экспериментатор должен нажимать кнопки в данной модели, чтобы *доказать*, что для каждого этажа (за исключением самого нижнего и самого верхнего) существуют три независимые двоичные переменные.

4. [*24*] Шаг E9 в сопрограмме работы лифта обычно отменяется на шаге E6; даже если он не отменен, он выполняет не очень большой объем работы. Объясните, при каких обстоятельствах лифт будет вести себя иначе, если шаг E9 удалить из модели. Например, может ли лифт иногда останавливаться на этажах в другом порядке?

5. [*20*] В табл. 1 человек 10 появляется на этаже 0 в момент 1048. Покажите, что, если бы человек 10 появился на этаже 2, а не на этаже 0, лифт отправился бы *вверх*, а не вниз после входа всех людей в кабину на этаже 1 несмотря на то, что человек 8 желает спуститься на этаж 0.

6. [*23*] В период 1183–1233 в табл. 1 пассажиры 7–9 входят в кабину лифта на этаже 1; лифт спускается на этаж 0, и из него выходит только пассажир 8. Затем лифт снова останавливается на этаже 1, предположительно для того, чтобы взять пассажиров 7 и 9, которые уже находятся в кабине лифта; таким образом, на этаже 1 лифт уже никто не ожидает. (Эта ситуация возникает в Калтехе довольно редко; если вы вошли в лифт, движущийся в ненужном вам направлении, вам придется снова сделать дополнительную остановку на пути к исходному этажу.) Во многих других типах лифтов пассажиры 7 и 9 не были бы взяты в лифт в момент 1183, так как индикаторы снаружи лифта показали бы, что он движется вниз, а не вверх. Им пришлось бы ожидать возвращения лифта. В описанной модели такие индикаторы не предусмотрены, а потому невозможно сказать, в каком направлении будет двигаться лифт, до тех пор, пока человек не попадет в кабину. Следовательно, табл 1 отражает реальную ситуацию.

Какие изменения необходимо внести в сопрограммы U и E, чтобы в описанной выше модели лифта использовались внешние индикаторы, благодаря которым людям не приходилось бы входить в кабину лифта, следующего в противоположном направлении?

7. [*25*] Часто программистам очень стыдно признавать свои ошибки в программах, но, если ошибки поучительны, о них не следует забывать; лучше сообщить о приобретенном опыте другим. При создании программы из этого раздела автор совершил (среди прочих) следующую ошибку. строка 154 содержала команду "`JANZ CYCLE`" вместо "`JANZ U4A`". Действительно, если лифт прибыл на этаж, нужный конкретному человеку, то больше нет необходимости "отменять" шаг U4. Поэтому можно просто перейти к процедуре `CYCLE` и продолжить моделирование других действий. В чем же заключается ошибка?

8. [*21*] Создайте код для выполнения шага E8, строки 277–292, который опущен в программе из данного раздела.

9. [*23*] Создайте код для подпрограммы `DECISION`, который опущен в программе из данного раздела.

10. [*40*] Вероятно, здесь стоит отметить, что, хотя автор пользовался лифтом в течение многих лет и думал, что довольно хорошо знает принцип его работы, только во время написания этого раздела он обнаружил новые факты об алгоритме выбора направления движения реального лифта. Шесть раз автору приходилось экспериментировать с лифтом, и всякий раз он думал, что уж теперь-то он окончательно понял его *modus operandi*. (Теперь автор не склонен к продолжению этих экспериментов, так как опасается, что во время нового эксперимента обнажится еще одна новая грань поведения лифта, которая будет противоречить предложенному здесь варианту алгоритма.) Мы часто склонны недооценивать степень непонимания предмета, но лишь до тех пор, пока не попытаемся промоделировать его с помощью компьютера.

Попробуйте описать действия некоторого хорошо знакомого вам лифта. Проверьте свой алгоритм с помощью экспериментов (подглядывать в его электрическую схему — нечестно!), а затем создайте дискретную модель этой системы и запустите ее на компьютере.

▸ **11.** [*21*] (*Фрагментарно обновляемая память.*) Следующая проблема часто возникает в задачах *синхронного* моделирования. Система имеет n переменных V[1], . . . , V[n], и на каждом шаге моделирования новые значения некоторых из них вычисляются на основе прежних. Предполагается, что вычисления выполняются "одновременно" в том смысле, что эти переменные не принимают новых значений до тех пор, пока не будут выполнены все операции присвоения. Таким образом, одновременное появление выражений

$$\texttt{V[1]} \leftarrow \texttt{V[2]} \qquad \text{и} \qquad \texttt{V[2]} \leftarrow \texttt{V[1]}$$

приведет к обмену значениями V[1] и V[2], это сильно отличается от того, что обычно происходит при последовательном вычислении.

Эту ситуацию, конечно же, можно промоделировать с помощью таблицы NEWV[1], . . . , NEWV[n]. Перед каждым шагом моделирования для этого следует установить NEWV[k] $\leftarrow$ V[k] для $1 \le k \le n$, затем записать все изменения величин V[k] в NEWV[k] и наконец установить V[k] $\leftarrow$ NEWV[k], $1 \le k \le n$. Но этот "очень прямолинейный" метод не очень подходит по следующим причинам: (1) часто n очень велико, а количество изменяемых переменных очень мало́, (2) часто переменные упорядочены не в виде таблицы V[1], . . . , V[n], а разбросаны в памяти самым произвольным образом, (3) этот метод не может обрабстать случай (что обычно является ошибкой), когда одна переменная принимает два значения на одном и том же шаге моделирования.

Предполагая, что количество изменяемых на одном шаге переменных очень мало́, придумайте эффективный алгоритм, который моделирует нужные действия, используя две вспомогательные таблицы NEWV[k] и LINK[k], $1 \le k \le n$. Если возможно, алгоритм должен обнаружить ошибку в случае, если одна переменная принимает два значения на одном и том же шаге.

▸ **12.** [*22*] Почему в описанной в этом разделе программе моделирования лучше использовать дважды связанные списки вместо однократно связанных или последовательных списков?

2.2.6. Массивы и ортогональные списки

Одним из простейших обобщений линейного списка является двумерный (или, в более общем случае, многомерный) массив данных. Например, рассмотрим следующую матрицу размера $m \times n$:

$$\begin{pmatrix} \texttt{A[1,1]} & \texttt{A[1,2]} & \ldots & \texttt{A[1,}n\texttt{]} \\ \texttt{A[2,1]} & \texttt{A[2,2]} & \ldots & \texttt{A[2,}n\texttt{]} \\ \vdots & \vdots & & \vdots \\ \texttt{A[}m\texttt{,1]} & \texttt{A[}m\texttt{,2]} & \ldots & \texttt{A[}m\texttt{,}n\texttt{]} \end{pmatrix}. \tag{1}$$

В этом двумерном массиве каждый узел A[j,k] принадлежит двум линейным спискам: списку "строки j" A[j,1], A[j,2], . . . , A[j,n] и списку "столбца k" A[1,k], A[2,k], . . . , A[m,k]. Такие ортогональные списки строк и столбцов и образуют двумерную структуру матрицы. Аналогичные замечания относятся и к многомерным массивам данных.

Последовательное распределение. Когда массив хранится по *последовательно* расположенным адресам, память обычно распределяется так, чтобы

$$\texttt{LOC(A[J,K])} = a_0 + a_1\texttt{J} + a_2\texttt{K}, \tag{2}$$

где a_0, a_1 и a_2 — константы. Рассмотрим более общий случай: предположим, что имеется четырехмерный массив элементов длиной в одно слово `Q[I,J,K,L]`, где $0 \le \texttt{I} \le 2$, $0 \le \texttt{J} \le 4$, $0 \le \texttt{K} \le 10$, $0 \le \texttt{L} \le 2$. Память следовало бы распределить таким образом, чтобы

$$\texttt{LOC(Q[I,J,K,L])} = a_0 + a_1\texttt{I} + a_2\texttt{J} + a_3\texttt{K} + a_4\texttt{L}. \tag{3}$$

Это значит, что изменение `I`, `J`, `K` или `L` сразу же приведет к вычислению изменения адреса элемента `Q[I,J,K,L]`. Наиболее естественный (и наиболее распространенный) способ распределения памяти заключается в упорядочении элементов согласно лексикографическому порядку их индексов (упр. 1.2.1–15(d)), который иногда называют "упорядочение по строкам":

`Q[0,0,0,0]`, `Q[0,0,0,1]`, `Q[0,0,0,2]`, `Q[0,0,1,0]`, `Q[0,0,1,1]`, ...,
`Q[0,0,10,2]`, `Q[0,1,0,0]`, ..., `Q[0,4,10,2]`, `Q[1,0,0,0]`, ...,
`Q[2,4,10,2]`.

Нетрудно видеть, что этот порядок удовлетворяет требованиям (3) и может быть выражен так:

$$\texttt{LOC(Q[I,J,K,L])} = \texttt{LOC(Q[0,0,0,0])} + 165\texttt{I} + 33\texttt{J} + 3\texttt{K} + \texttt{L}. \tag{4}$$

Вообще, k-мерный массив с элементами $\texttt{A}[\texttt{I}_1, \texttt{I}_2, \ldots, \texttt{I}_k]$ длиной c слов при

$$0 \le \texttt{I}_1 \le d_1, \quad 0 \le \texttt{I}_2 \le d_2, \quad \ldots, \quad 0 \le \texttt{I}_k \le d_k$$

может храниться в памяти в виде

$$\begin{aligned}&\texttt{LOC(A[I}_1\texttt{,I}_2\texttt{,}\ldots\texttt{,I}_k\texttt{])}\\ &\quad = \texttt{LOC(A[0,0,}\ldots\texttt{,0])} + c(d_2+1)\ldots(d_k+1)\texttt{I}_1 + \cdots + c(d_k+1)\texttt{I}_{k-1} + c\texttt{I}_k\\ &\quad = \texttt{LOC(A[0,0,}\ldots\texttt{,0])} + \sum_{1\le r\le k} a_r\texttt{I}_r,\end{aligned} \tag{5}$$

где

$$a_r = c\prod_{r<s\le k}(d_s+1). \tag{6}$$

Для доказательства этой формулы заметим, что a_r — это объем памяти, необходимой для хранения части массива $\texttt{A}[\texttt{I}_1, \ldots, \texttt{I}_r, \texttt{J}_{r+1}, \ldots, \texttt{J}_k]$, где $\texttt{I}_1, \ldots, \texttt{I}_r$ — константы, а $\texttt{J}_{r+1}, \ldots, \texttt{J}_k$ изменяются для всех $0 \le \texttt{J}_{r+1} \le d_{r+1}, \ldots, 0 \le \texttt{J}_k \le d_k$. Следовательно, по определению лексикографического порядка адрес $\texttt{A}[\texttt{I}_1, \ldots, \texttt{I}_k]$ будет изменяться точно на эту величину при изменении $\texttt{I}_r$ на единицу.

Формулы (5) и (6) соответствуют значению числа $\texttt{I}_1\texttt{I}_2\ldots\texttt{I}_k$ в системе счисления со смешанным основанием (mixed-radix number system). Например, для массива `TIME[W,D,H,M,S]` с $0 \le \texttt{W} < 4$, $0 \le \texttt{D} < 7$, $0 \le \texttt{H} < 24$, $0 \le \texttt{M} < 60$ и $0 \le \texttt{S} < 60$ адрес элемента `TIME[W,D,H,M,S]` будет равен адресу `TIME[0,0,0,0,0]` плюс величина "`W` недель + `D` дней + `H` часов + `M` минут + `S` секунд" в секундах. Конечно, массив с

2 419 200 элементами может понадобиться только для очень экзотического приложения.

Традиционный метод хранения массивов обычно подходит только для массива с полностью прямоугольной структурой, в которой все элементы $\mathtt{A}[\mathtt{I}_1,\mathtt{I}_2,\ldots,\mathtt{I}_k]$ имеют индексы в независимых диапазонах $l_1 \le \mathtt{I}_1 \le u_1$, $l_2 \le \mathtt{I}_2 \le u_2$, $\ldots$, $l_k \le \mathtt{I}_k \le u_k$. В упр. 2 показано, как можно адаптировать формулы (5) и (6), когда нижние границы $(l_1, l_2, \ldots, l_k)$ не равны $(0, 0, \ldots, 0)$.

Однако во многих случаях массив не является прямоугольным. Чаще всего он представляет собой *треугольную матрицу*, в которой хранятся только элементы $\mathtt{A}[j,k]$, например, для $0 \le k \le j \le n$:

$$\begin{pmatrix} \mathtt{A}[0,0] & & & \\ \mathtt{A}[1,0] & \mathtt{A}[1,1] & & \\ \vdots & \vdots & \ddots & \\ \mathtt{A}[n,0] & \mathtt{A}[n,1] & \ldots & \mathtt{A}[n,n] \end{pmatrix}. \tag{7}$$

Как правило, известно, что все остальные элементы матрицы равны нулю или $\mathtt{A}[j,k] = \mathtt{A}[k,j]$, так что достаточно хранить в памяти всего лишь около половины всех элементов матрицы. Для хранения нижней треугольной матрицы (7) в $\frac{1}{2}(n+1)(n+2)$ последовательных позициях памяти следует отказаться от линейного распределения памяти (2) и использовать распределение в виде

$$\mathtt{LOC(A[J,K])} = a_0 + f_1(\mathtt{J}) + f_2(\mathtt{K}), \tag{8}$$

где f_1 и f_2 — функции одной переменной. (Константа a_0 может быть включена в функцию f_1 или f_2.) Если способ адресации имеет вид (8), доступ к произвольному элементу $\mathtt{A}[j,k]$ можно достаточно легко осуществить с помощью двух (очень коротких) вспомогательных таблиц со значениями f_1 и f_2. К тому же эти функции потребуется вычислить только один раз.

Причем лексикографический порядок индексов массива (7) удовлетворяет условию (8), а для элементов длиной в одно слово получим простую формулу

$$\mathtt{LOC(A[J,K])} = \mathtt{LOC(A[0,0])} + \frac{\mathtt{J}(\mathtt{J}+1)}{2} + \mathtt{K}. \tag{9}$$

Однако существует более эффективный способ хранения треугольных матриц одного и того же размера. Предположим, что нужно сохранить две матрицы $\mathtt{A}[j,k]$ и $\mathtt{B}[j,k]$, где $0 \le k \le j \le n$. Тогда их можно свести к одной матрице $\mathtt{C}[j,k]$, где $0 \le j \le n$, $0 \le k \le n+1$, используя условие

$$\mathtt{A}[j,k] = \mathtt{C}[j,k], \qquad \mathtt{B}[j,k] = \mathtt{C}[k,j+1]. \tag{10}$$

Таким образом, получим

$$\begin{pmatrix} \mathtt{C}[0,0] & \mathtt{C}[0,1] & \mathtt{C}[0,2] & \ldots & \mathtt{C}[0,n+1] \\ \mathtt{C}[1,0] & \mathtt{C}[1,1] & \mathtt{C}[1,2] & \ldots & \mathtt{C}[1,n+1] \\ \vdots & & & & \vdots \\ \mathtt{C}[n,0] & \mathtt{C}[n,1] & \mathtt{C}[n,2] & \ldots & \mathtt{C}[n,n+1] \end{pmatrix} \equiv \begin{pmatrix} \mathtt{A}[0,0] & \mathtt{B}[0,0] & \mathtt{B}[1,0] & \ldots & \mathtt{B}[n,0] \\ \mathtt{A}[1,0] & \mathtt{A}[1,1] & \mathtt{B}[1,1] & \ldots & \mathtt{B}[n,1] \\ \vdots & & & & \vdots \\ \mathtt{A}[n,0] & \mathtt{A}[n,1] & \mathtt{A}[n,2] & \ldots & \mathtt{B}[n,n] \end{pmatrix}.$$

Так, две треугольные матрицы могут быть компактно упакованы в $(n+1)(n+2)$ ячейках памяти с линейной адресацией типа (2).

Обобщение треугольных матриц для больших измерений называется *тетраэдральным массивом* (*tetrahedral array*). Рассмотрение этой интересной темы продолжается в упр. 6–8.

Типичные приемы программирования при работе с последовательно адресуемыми массивами описаны в упр. 1.3.2–10 и в двух ответах к данному упражнению. Особенно интересными в этих программах являются фундаментальные методы эффективного обхода строк и столбцов, а также использование последовательных стеков.

Связанное распределение. Связанное распределение памяти прекрасно подходит и для представления многомерных массивов данных. Вообще, узлы могут содержать k полей связи, по одному для каждого списка, которому принадлежит этот узел. Связанное распределение памяти обычно используется в случаях, когда массивы данных не строго прямоугольны.

В качестве примера рассмотрим список, в котором каждый узел представляет описание некоторого человека, с четырьмя полями связи для обозначения: пола SEX, возраста AGE, цвета глаз EYES и цвета волос HAIR. Например с помощью полей EYES связываются узлы с одним цветом глаз и т. д. (рис. 13). Тогда нетрудно представить эффективный алгоритм вставки в этот список узлов с описаниями новых людей. Операция удаления в такой структуре будет выполняться гораздо медленнее, но ее эффективность можно повысить, используя дважды связанные списки. В таком случае можно представить алгоритмы с разной степенью эффективности для выполнения, например, таких запросов: "Найти всех голубоглазых блондинок в возрасте от 21 до 23 лет" (см. упр. 9 и 10). Подобного рода задачи, в которых узлы одного списка могут принадлежать нескольким другим спискам, встречаются довольно часто. Действительно, описанная в предыдущем разделе модель работы лифта содержит узлы, которые находятся сразу в двух списках: QUEUE и WAIT.

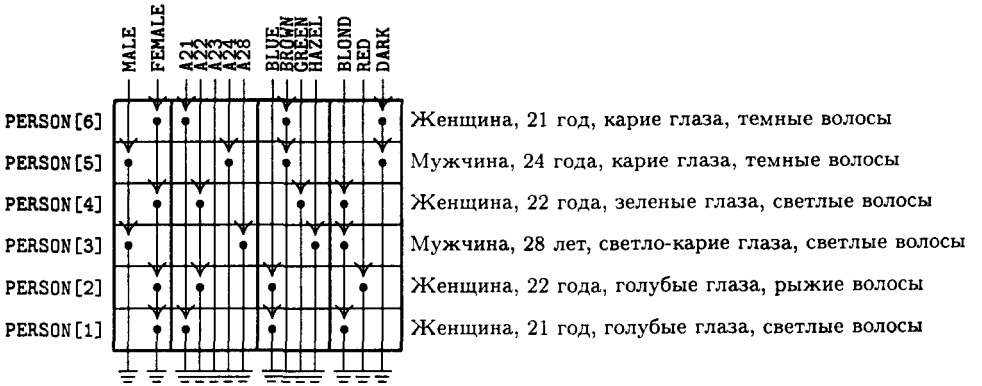
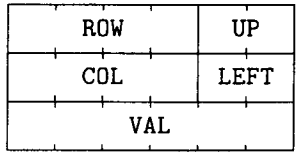


Рис. 13. Каждый узел находится в четырех различных списках.

В качестве примера использования связанного распределения памяти для прямоугольных списков рассмотрим *разреженные матрицы* (*sparse matrices*) (т. е. матрицы большого размера, в которых большинство элементов равно нулю). Назначение задачи — организовать работу с такими структурами, как с обычной матрицей,

но достичь при этом большой экономии времени и пространства в памяти за счет исключения из нее равных нулю элементов. Один способ организации произвольного доступа к элементам этой матрицы заключается в использовании методов хранения и извлечения данных, которые описаны в главе 6, т. е. поиск элемента A[j,k] выполняется на основе ключа "[j,k]". Существует, однако, еще один способ работы с разреженными матрицами, который часто более предпочтителен, поскольку он точнее соответствует структуре матрицы. Именно этот метод будет рассмотрен ниже.

Рассматриваемое здесь представление состоит из циклически связанных списков, т. е. циклически связанных строк и столбцов. Каждый узел матрицы имеет длину в три слова и содержит пять полей:



(11)

Здесь ROW и COL — индексы, обозначающие строку и столбец узла; VAL — значение, хранимое в элементе матрицы; LEFT и UP — связи со следующим ненулевым элементом слева в строке или сверху в столбце соответственно. Для каждой строки и каждого столбца заданы заголовки списков BASEROW[i] и BASECOL[j] соответственно. Эти узлы идентифицируются благодаря следующим условиям:

$$\texttt{COL(LOC(BASEROW[}i\texttt{]))} < 0 \qquad \text{и} \qquad \texttt{ROW(LOC(BASECOL[}j\texttt{]))} < 0.$$

Как обычно, в циклическом списке ссылка LEFT в строке BASEROW[i] является адресом крайнего справа значения в этой строке, а ссылка UP в строке BASECOL[j] указывает на последнее значение в столбце. Например, матрица

$$\begin{pmatrix} 50 & 0 & 0 & 0 \\ 10 & 0 & 20 & 0 \\ 0 & 0 & 0 & 0 \\ -30 & 0 & -60 & 5 \end{pmatrix} \tag{12}$$

при таких условиях будет иметь вид, приведенный на рис. 14.

При использовании метода последовательного распределения памяти для матрицы размера 200 × 200 потребуется 40 000 слов, что гораздо больше размера оперативной памяти большинства компьютеров. Но для представления умеренно разреженной матрицы размера 200 × 200 с помощью описанного выше способа в оперативной памяти компьютера MIX потребуется только 4 000 слов (см. упр. 11). При этом время доступа к произвольному элементу A[j,k] также будет вполне приемлемым, *если* в каждой строке и каждом столбце содержится немного ненулевых (или неодинаковых) элементов. Поскольку в большинстве алгоритмов обработки матриц используется последовательный доступ к элементам, а не произвольный, то при таком связанном представлении данных работа может выполняться гораздо быстрее, чем при последовательном представлении.

Типичным примером нетривиального алгоритма работы с разреженными матрицами такого типа является *осевое преобразование* (*pivot step*), которое является

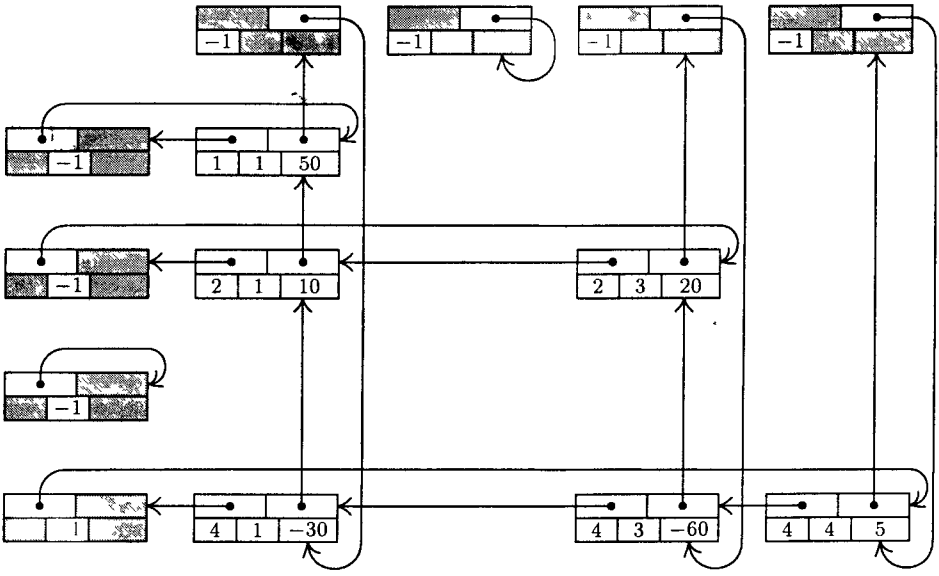


Рис. 14. Представление матрицы (12) с узлами в виде

LEFT	UP
ROW	COL	VAL

Заголовки списков находятся слева и сверху

важнейшей частью алгоритмов для решения линейных уравнений, обращения матриц и решения задач линейного программирования с помощью симплекс-метода. Осевое преобразование представляет собой следующее преобразование матрицы.

$$\begin{array}{l} \\ \text{Осевая строка} \\ \\ \text{Любая другая строка} \\ \\ \end{array} \begin{array}{c} \text{До преобразования} \\ \begin{array}{cc} \text{Осевой столбец} & \text{Любой другой столбец} \end{array} \\ \begin{pmatrix} \vdots & \vdots \\ \cdots\ a & \cdots\ b\ \cdots \\ \vdots & \vdots \\ \cdots\ c & \cdots\ d\ \cdots \\ \vdots & \vdots \end{pmatrix} \end{array} , \quad \begin{array}{c} \text{После преобразования} \\ \begin{array}{cc} \text{Осевой столбец} & \text{Любой другой столбец} \end{array} \\ \begin{pmatrix} \vdots & \vdots \\ \cdots\ 1/a & \cdots\ b/a\ \cdots \\ \vdots & \vdots \\ \cdots\ -c/a & \cdots\ d-bc/a\ \cdots \\ \vdots & \vdots \end{pmatrix} \end{array} \tag{13}$$

Предполагается, что *осевой элемент*, a, не равен нулю. Например, применение осевого преобразования к матрице (12), где осевым элементом считается число 10 в строке 2 и столбце 1, приводит к такому результату:

$$\begin{pmatrix} -5 & 0 & -100 & 0 \\ 0.1 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 \\ 3 & 0 & 0 & 5 \end{pmatrix}. \tag{14}$$

Цель данного исследования заключается в создании алгоритма осевой операции для разреженной матрицы, подобной матрице, представленной на рис. 14. Ясно, что преобразование типа (13) повлияет только на те строки матрицы, в которых есть ненулевые элементы в осевом столбце, и только на те столбцы матрицы, в которых есть ненулевые элементы в осевой строке.

Алгоритм поворота во многом напоминает прямолинейное применение методов связывания, которые рассматривались выше. В частности, он во многом похож на алгоритм 2.2.4A сложения полиномов. Однако существуют дополнительные соображения, которые немного усложняют нашу задачу: если в (13) $b \neq 0$, $c \neq 0$, но $d = 0$, в представлении разреженной матрицы не будет элемента d, то его потребуется вставить; а если $b \neq 0$, $c \neq 0$, $d \neq 0$, но $d - bc/a = 0$, то придется удалить элемент, который там находился прежде. Эти операции вставки и удаления более интересны при работе с двумерным массивом, чем с одномерным. Для их выполнения необходимо знать обо всех связях, вовлеченных в данный процесс. Алгоритм обрабатывает все строки матрицы последовательно снизу вверх. Для эффективного выполнения операций вставки и удаления необходимо ввести таблицу указательных переменных PTR[j], по одной для каждого рассматриваемого столбца. С помощью этих переменных совершается обход столбцов по направлению снизу вверх, в результате чего предоставляется возможность обновления соответствующих связей в обоих измерениях.

Алгоритм S (*Осевое преобразование разреженной матрицы*). Выполним операцию осевого преобразования (13) для матрицы, показанной на рис. 14. Предположим, что PIVOT — это переменная связи, которая указывает на осевой элемент. В алгоритме используется вспомогательная таблица переменных связи PTR[j], по одной для каждого столбца матрицы. Предполагается, что переменная ALPHA и поле VAL для каждого узла имеют тип числа с плавающей запятой или рационального числа, а все остальные — целочисленный тип.

S1. [Инициализация.] Установить ALPHA $\leftarrow$ 1.0/VAL(PIVOT), VAL(PIVOT) $\leftarrow$ 1.0 и

$$\texttt{I0} \leftarrow \texttt{ROW(PIVOT)}, \quad \texttt{P0} \leftarrow \texttt{LOC(BASEROW[I0])};$$
$$\texttt{J0} \leftarrow \texttt{COL(PIVOT)}, \quad \texttt{Q0} \leftarrow \texttt{LOC(BASECOL[J0])}.$$

S2. [Обработка осевой строки.] Установить P0 $\leftarrow$ LEFT(P0), J $\leftarrow$ COL(P0). Если J < 0, перейти к шагу S3 (осевая строка пройдена). В противном случае установить PTR[J] $\leftarrow$ LOC(BASECOL[J]), VAL(P0) $\leftarrow$ ALPHA $\times$ VAL(P0) и повторить шаг S2.

S3. [Поиск новой строки.] Установить Q0 $\leftarrow$ UP(Q0). (В остальной части алгоритма последовательно снизу вверх перебираются все строки, которые содержат элемент данных в осевом столбце.) Установить I $\leftarrow$ ROW(Q0). Если I < 0, выполнение алгоритма прекращается. Если I = I0, повторить шаг S3 (осевая строка обработана). В противном случае установить P $\leftarrow$ LOC(BASEROW[I]), P1 $\leftarrow$ LEFT(P). (Указатели P и P1 позволяют совершить проход по строке I справа налево так же, как P0 позволяет это сделать для строки I0. Алгоритм 2.2.4A выполняется аналогично. При этом P0 = LOC(BASEROW[I0]).)

S4. [Поиск нового столбца.] Установить P0 $\leftarrow$ LEFT(P0), J $\leftarrow$ COL(P0). Если J < 0, установить VAL(Q0) $\leftarrow -$ALPHA $\times$ VAL(Q0) и вернуться к шагу S3. Если J = J0,

повторить шаг S4. (Таким образом, элемент осевого столбца в строке I обрабатывается *после* всех элементов других столбцов; причина заключается в том, что значение VAL(QQ) потребуется на шаге S7.)

S5. [Поиск элемента I, J.] Если COL(P1) > J, установить P ← P1, P1 ← LEFT(P) и повторить шаг S5. Если COL(P1) = J, перейти к шагу S7. В противном случае перейти к шагу S6 (вставить новый элемент в столбце J строки I).

S6. [Вставка элемента I, J.] Если ROW(UP(PTR[J])) > I, установить PTR[J] ← UP(PTR[J]) и повторить шаг S6. (Иначе получим ROW(UP(PTR[J])) < I; новый элемент нужно вставить сразу над узлом NODE(PTR[J]) в вертикальном направлении и слева от узла NODE(P) в горизонтальном направлении.) В противном случае установить X ⇐ AVAIL, VAL(X) ← 0, ROW(X) ← I, COL(X) ← J, LEFT(X) ← P1, UP(X) ← UP(PTR[J]), LEFT(P) ← X, UP(PTR[J]) ← X, P1 ← X.

S7. [Осевое преобразование.] Установить VAL(P1) ← VAL(P1) − VAL(Q0) × VAL(P0). Если теперь VAL(P1) = 0, следует перейти к шагу S8. (*Замечание.* При использовании системы счисления с плавающей запятой условие "VAL(P1) = 0" следует заменить условием "|VAL(P1)| < EPSILON" или, что еще лучше, условием "большинство значащих цифр VAL(P1) утрачено при вычитании".) В противном случае установить PTR[J] ← P1, P ← P1, P1 ← LEFT(P) и вернуться к шагу S4.

S8. [Удаление элемента I, J.] Если UP(PTR[J]) ≠ P1 (или, что, по сути, то же самое, если ROW(UP(PTR[J])) > I), установить PTR[J] ← UP(PTR[J]) и повторить шаг S8. В противном случае установить UP(PTR[J]) ← UP(P1), LEFT(P) ← LEFT(P1), AVAIL ⇐ P1, P1 ← LEFT(P). Вернуться к шагу S4. ▮

Читателю предлагается (в качестве очень поучительного упражнения) самостоятельно создать программу для реализации этого алгоритма (см. упр. 15). Здесь же стоит отметить, что для каждого узла BASEROW[i] и BASECOL[j] достаточно только одного слова памяти, поскольку большинство их полей не будет востребовано (см. заштрихованные области на рис. 14, а также программу из раздела 2.2.5.) Более того, в целях дополнительной экономии памяти значение −PTR[j] можно хранить как ROW(LOC(BASECOL[j])). Время выполнения алгоритма S приблизительно пропорционально количеству матричных элементов, которые вовлечены в операцию осевого преобразования.

Такое представление разреженных матриц с помощью ортогональных циклических списков очень поучительно, но специалисты по численному анализу разработали несколько более совершенных методов. [См., например, работу Fred G. Gustavson, *ACM Trans. on Math. Software* **4** (1978), 250–269; а также алгоритмы работы с графами и задачами сетевого планирования в главе 7.]

УПРАЖНЕНИЯ

1. [*17*] Предложите формулу для LOC(A[J,K]), если A — это матрица типа (1), а каждый ее узел состоит из двух слов, причем все узлы хранятся последовательно в лексикографическом порядке индексов.

▸ **2.** [*21*] Формула (6) выведена на основании предположения, что $0 \le I_r \le d_r$ для $1 \le r \le k$. Найдите общую формулу, в которой предполагается, что $l_r \le I \le u_r$, где l_r и u_r — любые значения нижней и верхней границ данного измерения

3. [*21*] В этом разделе рассматривались нижние треугольные матрицы `A[`j`,`k`]`, где $0 \le k \le j \le n$. Как следует видоизменить приведенные рассуждения в случае, если отсчет индексов начинается с единицы, а не с нуля и $1 \le k \le j \le n$?

4. [*22*] Покажите, что при хранении *верхней* треугольной матрицы `A[`j`,`k`]`, где $0 \le j \le k \le n$, в лексикографическом порядке индексов распределение памяти будет удовлетворять условию (8). Найдите в таком случае формулу для `LOC(A[J,K])`.

5. [*20*] Покажите, что значение `A[J,K]` можно занести в регистр A компьютера `MIX`, выполнив одну команду и использовав инструменты косвенной адресации, которые описываются в упр. 2.2.2–3, даже если `A` является *треугольной* матрицей типа (9). (Предполагается, что `J` и `K` находятся в индексных регистрах.)

▶ **6.** [*M24*] Рассмотрим "тетраэдрические массивы" `A[`i,j,k`]`, `B[`i,j,k`]`, где $0 \le k \le j \le i \le n$ в массиве `A` и $0 \le i \le j \le k \le n$ в массиве `B`. Предположим, что оба эти массива хранятся в последовательных адресах памяти в лексикографическом порядке индексов. Покажите, что `LOC(A[I,J,K])` $= a_0 + f_1$`(I)` $+ f_2$`(J)` $+ f_3$`(K)` для некоторых функций f_1, f_2, f_3. Можно ли получить аналогичное выражение для `LOC(B[I,J,K])`?

7. [*M23*] Найдите общую формулу распределения памяти для k-мерного тетраэдрического массива `A[`$i_1,i_2,\ldots,i_k$`]`, где $0 \le i_k \le \cdots \le i_2 \le i_1 \le n$.

8. [*33*] (Задача П. Вегнера .) Предположим, что в памяти необходимо разместить шесть тетраэдрических массивов `A[I,J,K]`, `B[I,J,K]`, `C[I,J,K]`, `D[I,J,K]`, `E[I,J,K]` и `F[I,J,K]`, где $0 \le$ `K` $\le$ `J` $\le$ `I` $\le n$. Существует ли какой-нибудь эффективный способ выполнения этой задачи, аналогичный (10), но для двумерного случая?

9. [*22*] Рассмотрим таблицу такого же типа, как и таблица на рис. 13, но гораздо бóльшую, в которой все связи направлены в одну сторону (а именно — выполняется условие `LINK(X)` $<$ `X` для всех узлов и ссылок). Придумайте алгоритм для поиска адресов всех голубоглазых блондинок в возрасте от 21 до 23 лет, основанный на обходе различных полей связей, причем таким образом, что по завершении выполнения алгоритма для каждого из списков выполняется по крайней мере один обход каждого из списков `FEMALE`, `A21`, `A22`, `A23`, `BLOND` и `BLUE`.

10. [*26*] Можно ли придумать более совершенный способ организации таблицы с персональными данными, чтобы описанный в предыдущем примере поиск можно было выполнить более эффективным способом? (Простой ответ "Да" или "Нет" в данном случае *не* годится.)

11. [*11*] Допустим, что в каждой строке матрицы размера 200×200 находится по крайней мере четыре ненулевых элемента Какой объем памяти потребуется для представления ее в виде, показанном на рис. 14, если для каждого узла используется по три слова, а для заголовков списков — по одному?

▶ **12.** [*20*] Чему равны `VAL(QO)`, `VAL(PO)` и `VAL(P1)` в начале шага S7, если их выразить с помощью переменных a, b, c, d из (13)?

▶ **13.** [*22*] Почему в матрице на рис. 14 циклические списки используются вместо линейных списков? Можно ли изменить алгоритм S так, чтобы в нем не использовалась циклическая связь?

14. [*22*] Алгоритм S позволяет сэкономить время выполнения осевого преобразования разреженной матрицы, поскольку он предоставляет возможность пропускать столбцы, в которых значение элемента из осевой строки равно нулю Покажите, что такая экономия может быть получена в большой разреженной матрице, которая хранится в последовательном порядке, за счет применения вспомогательной таблицы `LINK[`j`]`, $1 \le j \le n$.

▸ **15.** [*29*] Создайте программу на языке MIXAL, которая реализует алгоритм S. Предположите, что поле VAL содержит числа с плавающей запятой и для работы с ними в компьютере MIX предусмотрено несколько команд: FADD, FSUB, FMUL и FDIV. Для простоты положим, что результатом команды сложения FADD и вычитания FSUB будет нуль, если при сложении или вычитании операндов будут утрачены старшие разряды. Поэтому на шаге S7 можно применить условие "VAL(P1) = 0". При работе операторов для чисел с плавающей запятой регистр rA используется, а регистр rX — нет.

16. [*25*] Создайте алгоритм *копирования* разреженной матрицы. (Иначе говоря, напишите алгоритм получения в памяти двух различных представлений матрицы, которые имеют показанную на рис. 14 форму, при условии, что сначала задано только одно такое представление.)

17. [*26*] Создайте алгоритм *умножения* двух разреженных матриц A и B с образованием новой матрицы C, в которой $\mathtt{C}[i,j] = \sum_k \mathtt{A}[i,k]\mathtt{B}[k,j]$. Причем обе исходные матрицы и результирующая матрица должны иметь представленную на рис. 14 форму.

18. [*22*] Следующий алгоритм позволяет заменить исходную матрицу $\mathtt{A}[i,j]$, где $1 \le i,j \le n$, обратной ей матрицей.

i) *Для* $k = 1, 2, \ldots, n$ *выполнить следующие действия: просмотреть строку* k *в столбцах, которые еще не использовались в качестве осевых столбцов, и найти наибольший по абсолютной величине элемент; установить* $\mathtt{C}[k]$ *равным номеру столбца, в котором элемент был найден, и выполнить осевое преобразование, используя этот элемент как осевой.* (Если все такие элементы равны нулю, то матрица является сингулярной и не имеет обратной.)

ii) Переставить строки и столбцы так, чтобы k-я строка стала строкой $\mathtt{C}[k]$, а столбец $\mathtt{C}[k]$ стал k-м.

Используя описанный выше алгоритм, найдите вручную обратную матрицу для матрицы

$$\begin{pmatrix} 1 & 2 & 3 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{pmatrix}.$$

19. [*31*] Измените описанный в упр. 18 алгоритм так, чтобы для разреженной матрицы, подобной представленной на рис. 14, можно было получить обратную матрицу. Уделите особое внимание эффективности выполнения перестановок среди строк и среди столбцов на шаге (ii).

20. [*20*] Имеется *трехдиагональная матрица* (*tridiagonal matrix*), в которой элементы a_{ij} не равны нулю только тогда, когда $|i-j| \le 1$ для $1 \le i,j \le n$. Покажите, что для нее существует такая функция размещения

$$\mathtt{LOC(A[I,J])} = a_0 + a_1\mathtt{I} + a_2\mathtt{J}, \quad |\mathtt{I}-\mathtt{J}| \le 1,$$

которая позволяет представить все ее ненулевые элементы в $(3n-2)$ последовательно расположенных ячейках.

21. [*20*] Предложите функцию размещения для матрицы размера $n \times n$, где n является переменной. Элементы матрицы $\mathtt{A}[\mathtt{I},\mathtt{J}]$ для $1 \le \mathtt{I},\mathtt{J} \le n$ должны занимать n^2 последовательно расположенных ячеек независимо от величины n.

22. [*M25*] (Задача П. Чоула , 1961.) Найдите такой полином $p(i_1,\ldots,i_k)$, который принимает каждое неотрицательное целое значение в точности один раз при обходе индексов $(i_1,\ldots,i_k)$ для всех k-мерных неотрицательных целых векторов с учетом того, что из условия $i_1+\cdots+i_k < j_1+\cdots+j_k$ следует $p(i_1,\ldots,i_k) < p(j_1,\ldots,j_k)$.

23. [*23*] *Расширяемая матрица* (*extendible matrix*) с исходными размерами 1×1 растет от размера $m \times n$ к размеру $(m+1) \times n$ или к размеру $m \times (n+1)$ за счет добавления

новой строки или нового столбца Найдите для нее простую функцию распределения, согласно которой элементы A[I,J] занимают mn последовательных ячеек, где $0 \le \mathtt{I} < m$ и $0 \le \mathtt{J} < n$, причем при росте матрицы ее элементы не изменяют своего положения в памяти

▶ **24.** [*25*] (*Еще одна хитрость при работе с разреженными массивами*) Предположим, что существует некий неинициализированный массив огромного размера и необходимо обеспечить доступ к его немногим произвольным элементам При первой попытке доступа к элементу A[k] для него следует установить значение 0, причем смысл данной уловки заключается в том, чтобы избежать инициализации нулями сразу всех элементов массива и исключить связанные с этим большие затраты времени Предложите надежный способ чтения и записи любых элементов A[k] для заданного k, ничего не зная о фактическом начальном содержимом памяти и выполняя лишь небольшое фиксированное количество операций доступа к этому массиву

2.3. ДЕРЕВЬЯ

ПРИСТУПИМ ТЕПЕРЬ к изучению деревьев — наиболее важных нелинейных структур, которые встречаются при работе с компьютерными алгоритмами. Вообще говоря, древовидная структура задает для узлов отношение "ветвления", которое во многом напоминает строение обычного дерева.

Формально *дерево* (*tree*) определяется как конечное множество T одного или более узлов со следующими свойствами:

a) существует один выделенный узел, а именно — *корень* (*root*) данного дерева T;
b) остальные узлы (за исключением корня) распределены среди $m \geq 0$ непересекающихся множеств $T_1, \ldots, T_m$, и каждое из этих множеств, в свою очередь, является деревом; деревья $T_1, \ldots, T_m$ называются *поддеревьями* (*subtrees*) данного корня.

Как видите, это определение является рекурсивным: дерево определено на основе понятия дерева. Однако никакого порочного круга определений здесь нет, так как состоящее из одного узла дерево содержит только корень, а все остальные деревья с $n > 1$ узлами определяются на основе деревьев с n узлами. Следовательно, это позволяет дать определение дерева с двумя, тремя и более узлами. Помимо рекурсивного способа определения, существует несколько нерекурсивных способов определения деревьев (например, см. упр. 10, 12 и 14, а также раздел 2.3.4), но рекурсивное определение наиболее приемлемо, так как рекурсивность отражает неотъемлемое свойство всех древовидных структур. Рекурсивный характер деревьев можно наблюдать и в природе, например почки молодых деревьев растут и со временем превращаются в ветви (поддеревья), на которых снова появляются почки, которые также растут и со временем превращаются в ветви (поддеревья), и т. д. В упр. 3 методом индукции по числу узлов дерева доказано несколько важных свойств деревьев на основе приведенного выше рекурсивного определения.

Из этого определения следует, что каждый узел дерева является корнем некоторого поддерева данного дерева. Количество поддеревьев узла называется *степенью* (*degree*) этого узла. Узел со степенью нуль называется *концевым узлом* (*terminal node*) или *листом* (*leaf*). Неконцевой узел называется *узлом ветвления* (*branch node*). *Уровень* (*level*) узла по отношению к дереву T определяется рекурсивно следующим образом. Уровень корня дерева T равен нулю, а уровень любого другого узла на единицу выше, чем уровень корня ближайшего поддерева дерева T, содержащего данный узел.

Эти понятия иллюстрируются на рис. 15 на примере дерева с семью узлами. Узел A является корнем, который имеет два поддерева: $\{B\}$ и $\{C, D, E, F, G\}$. Корнем дерева $\{C, D, E, F, G\}$ является узел C. Уровень узла C равен 1 по отношению ко всему дереву. Он имеет три поддерева, $\{D\}$, $\{E\}$ и $\{F, G\}$, поэтому C имеет степень 3. Концевыми на рис. 15 являются узлы B, D, E и G. Узел F — единственный узел со степенью 1, а узел G — единственный узел со степенью 3.

Если в п. (b) данного выше определения имеет значение относительный порядок поддеревьев $T_1, \ldots, T_m$, то дерево является *упорядоченным* (*ordered tree*). Если в упорядоченном дереве $m \geq 2$, то имеет смысл назвать поддерево T_2 вторым поддеревом данного корня и т. д. Упорядоченные деревья иногда также называются плоскими деревьями (plane trees), поскольку при их упорядочении имеет значение

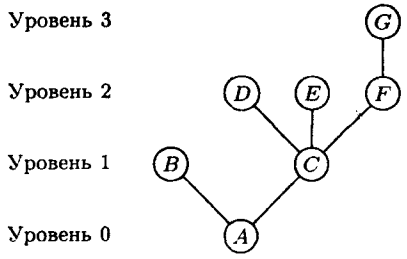


Рис. 15. Пример дерева.

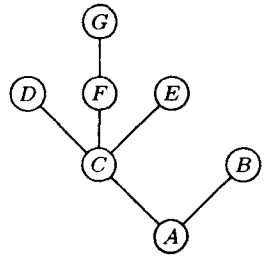


Рис. 16. Еще один пример дерева.

способ размещения дерева на плоскости. Если не считать различными два дерева, которые отличаются только относительным порядком поддеревьев узлов, то дерево называется *ориентированным* (*oriented*), поскольку при этом имеет значение только относительная ориентация узлов, а не их порядок. Сама природа представления данных в компьютере определяет неявный порядок любого дерева, поэтому в большинстве случаев упорядоченные деревья представляют наибольший интерес. Далее будем неявно предполагать, что *все рассматриваемые деревья являются упорядоченными, если явно не указано обратное.* Соответственно деревья на рис. 15 и 16 в общем случае рассматриваются как разные, хотя как ориентированные деревья они совершенно одинаковы.

Лес (*forest*) — это множество (обычно упорядоченное), не содержащее ни одного непересекающегося дерева или содержащее несколько непересекающихся деревьев. Тогда еще одна формулировка п. (b) в данном выше определении дерева могла бы выглядеть так: *узлы дерева при условии исключения корня образуют лес.*

Между абстрактными понятиями леса и деревьев существует не очень заметная разница. При удалении корня дерева получим лес, и наоборот: при добавлении одного узла в лес, все деревья которого рассматриваются как поддеревья нового узла, получим дерево. Поэтому понятия "лес" и "дерево" часто используются как равнозначные при работе со структурами данных.

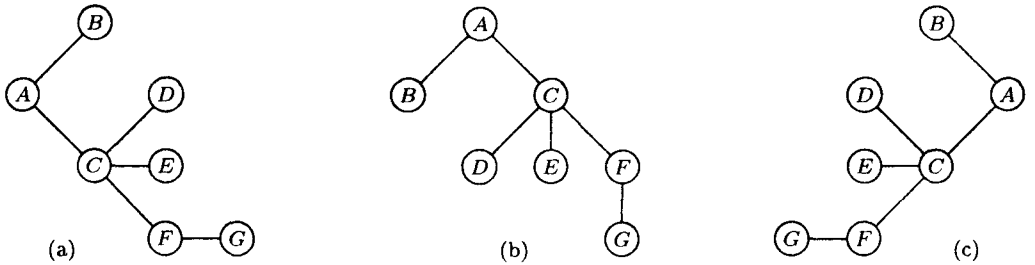


Рис. 17. Как следует рисовать деревья?

Графически деревья можно представить по-разному. Например, для дерева на рис. 15 существует еще три принципиально отличных альтернативных варианта, которые, как показано на рис. 17, отличаются расположением узлов относительно корня. Правила схематического изображения древовидных структур — это не чья-то прихоть, так как довольно часто при работе с ними приходится говорить о том, что один узел находится "над" другим, "выше" другого или является "крайним

справа и т. д. Одни алгоритмы обработки древовидных структур называются нисходящими, а другие — восходящими. Без строгого соглашения о правилах схематического изображения деревьев такая терминология может привести к путанице.

Может показаться, что схема, представленная на рис. 15, выглядит предпочтительнее просто потому, что именно так растут деревья в природе. При отсутствии любых других существенных причин для более предпочтительного выбора следует использовать уже освященную веками традицию природы. Имея это в виду, автор при работе над данной серией книг следовал правилу изображения деревьев с корнем внизу, но после двух лет работы обнаружилась ошибочность такого выбора. Изучение литературы и многочисленные неформальные обсуждения широкого круга алгоритмов со специалистами в области информатики показали, что в более чем 80% рассмотренных случаев деревья изображаются *с корнем вверху*. Это не более чем непреодолимая тенденция рисовать от руки по направлению сверху вниз, а не снизу вверх (что вполне объяснимо, если вспомнить, как мы пишем). Даже термин "поддерево" в противоположность термину "наддерево" ассоциируется с нисходящим родством. Поэтому будем считать, что *рис. 15 просто перевернут "вверх ногами"*. Впредь почти всегда схема дерева будет выглядеть так, как на рис. 17, (b), т. е. с корнем сверху и листьями внизу. В соответствии с такой ориентацией назовем корневой узел *вершиной* (*apex*) дерева и будем характеризовать уровни узлов как *мелкие* и *глубокие*.

Для рассмотрения деревьев необходимо создать хорошую описательную терминологию. Вместо двусмысленных формулировок наподобие "над" и "под" лучше использовать общепринятые термины, которые применяются при работе с *генеалогическими деревьями*. На рис. 18 показаны два наиболее распространенных типа генеалогических деревьев. Они отличаются тем, что в *родословной* показаны предки конкретного человека, а в *родовой схеме* — его наследники.

При наличии "скрещивания" родословная уже не является деревом, поскольку разные ветви дерева (как было отмечено выше) не могут соединяться. Для устранения этого несоответствия на рис. 18, (a) королева Виктория и принц Альберт дважды упоминаются в шестом поколении, а король Кристиан IX и королева Луиза — дважды в пятом и шестом поколениях. Родословную следует рассматривать как истинное дерево, если каждый его узел представляет не просто отдельного человека, а человека в качестве матери или отца какого-то другого человека.

Стандартная терминология древовидных структур происходит от генеалогических деревьев *второго* типа, а именно — от родовой схемы. Каждый узел называется *родителем* (*parent*) корней его поддеревьев, а сами корни называются *братьями-сестрами* (*siblings*), а также *детьми* (*children*) своего родителя. Корень всего дерева не имеет родителя. Например, на рис. 19 узел C имеет трех детей, D, E и F, узел E является родителем узла G, а узлы B и C являются братьями-сестрами. Эту терминологию, очевидно, можно расширить. Например, узел A является прапрародителем (т. е. прадедушкой или прабабушкой) узла G, а узел B — двоюродным родителем (т. е. дядей или тетей) узла F, узлы H и F являются двоюродными братьями-сестрами. Одни авторы вместо терминов "родители", "дети", "сестры-братья" предпочитают использовать "мужскую" терминологию: "отец" (father), "сын" (son), "брат" (brother), а другие — женскую: "мать" (mother) , "дочь" (daughter), "сестра" (sister). В любом случае узел имеет не более одного родителя или

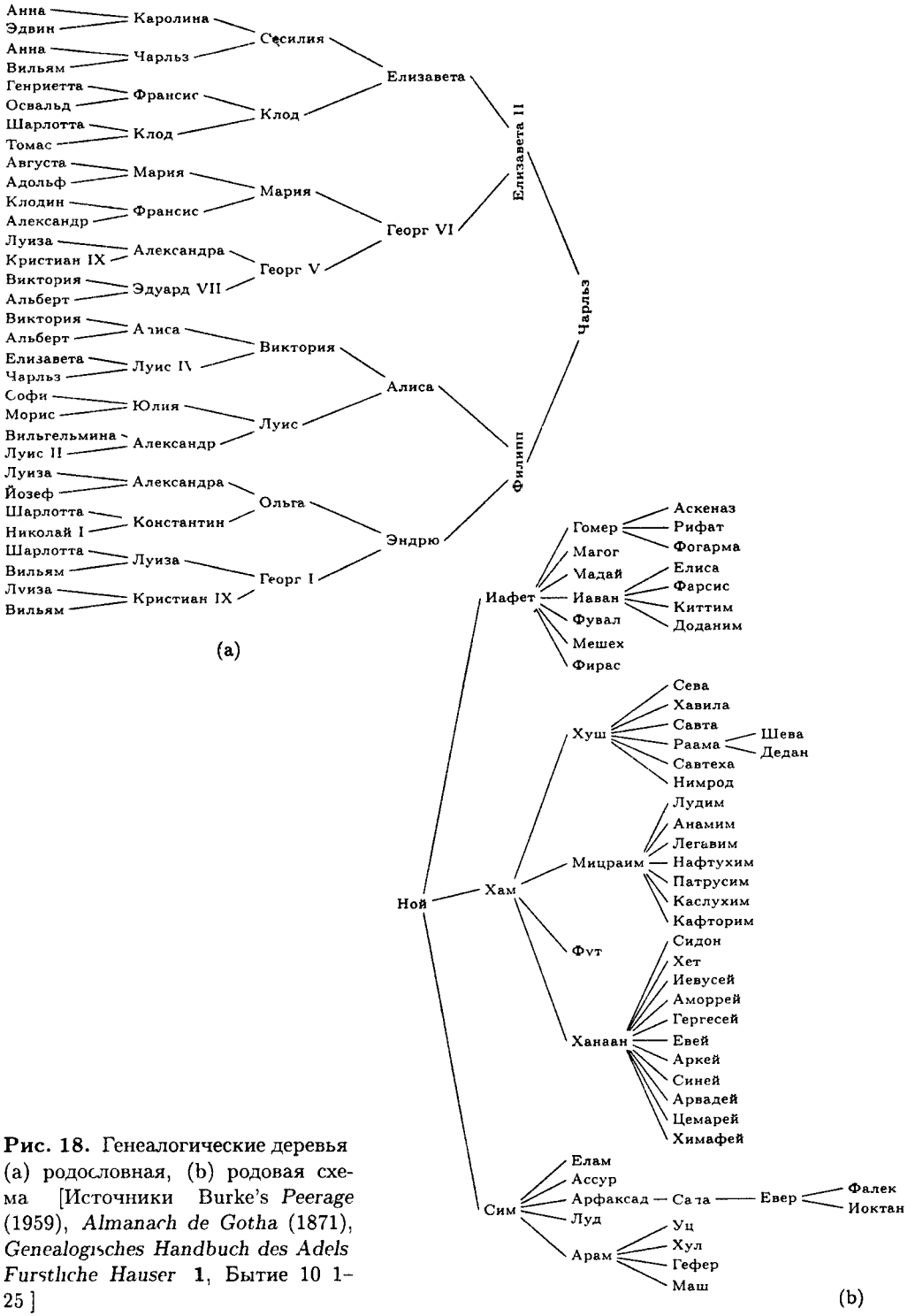


Рис. 18. Генеалогические деревья (a) родословная, (b) родовая схема [Источники Burke's *Peerage* (1959), *Almanach de Gotha* (1871), *Genealogisches Handbuch des Adels Furstliche Hauser* **1**, Бытие 10 1–25]

предка. Далее для обозначения родства, которое может простираться на несколько уровней дерева, будут использоваться термины "предок" (ancestor) и "потомок" (descendant). Например, потомками узла C на рис. 19 являются узлы D, E, F и G, а предками узла G — узлы E, C и A.

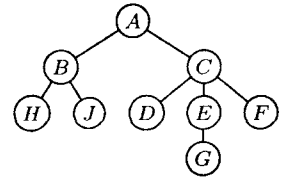
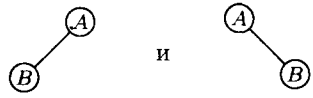


Рис. 19. Обычная схема дерева

Родословная, показанная на рис. 18, (a), является примером *бинарного дерева* (*binary tree*) — одного из наиболее важных типов деревьев. Читатель наверняка не раз встречался с бинарными деревьями в соревнованиях по теннису или других турнирах, которые проводятся по олимпийской системе с выбыванием проигравшего. Каждый узел бинарного дерева имеет не более двух поддеревьев, причем в случае только одного поддерева следует различать левое и правое. Строго говоря, бинарное дерево — это *конечное множество узлов, которое является пустым или состоит из корня и двух непересекающихся бинарных деревьев*, которые называются левым и правым поддеревьями данного корня.

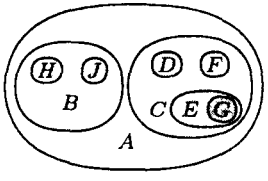
Тщательно изучим это рекурсивное определение бинарного дерева. Обратите внимание на то, что бинарное дерево *не* является особым типом ранее определенного обычного дерева. На самом деле это совершенно другое понятие (хотя впоследствии мы увидим, что у них есть много общего). Например, бинарные деревья



(1)

являются различными, так как в первом случае корень имеет пустое правое поддерево, а в другом — пустое левое поддерево. Однако как деревья они совершенно идентичны. Бинарное дерево может быть пустым, а обычное дерево — нет. Следовательно, при работе с бинарными деревьями нужно не забывать об обязательном эпитете "бинарный", чтобы не путать их с обычными деревьями. Некоторые авторы предлагают несколько иное определение бинарного дерева (см. упр. 20).

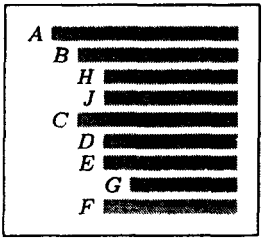
Древовидную структуру можно представить графически несколькими способами, которые могут выглядеть совершенно не так, как настоящие деревья в природе. На рис. 20 показаны три способа представления структуры с рис. 19: по сути, на рис. 20, (a) дерево с рис. 19 представлено в виде *ориентированного дерева* (*oriented tree*). Эта схема является частным случаем *вложенных множеств*(*nested sets*), а именно — набором множеств, в котором либо каждая пара множеств не пересекается, либо одно множество содержит другое (см. упр. 10). Если на рис. 20, (a) вложенные множества расположены в одной плоскости, то на рис. 20, (b) они находятся на одной линии, причем таким образом указывается и упорядочение данного дерева. Схема на рис. 20, (b) может рассматриваться как некая алгебраическая формула с вложенными скобками. Наконец, на рис. 20, (c) показан еще один общий способ представления древовидных структур с использованием *отступа* (*indentation*). Само по себе количество разных методов представления является прекрасным доказательством важности древовидных структур как в повседневной жизни, так и в программировании. В итоге любая классификация с иерархической структурой имеет древовидную структуру.



(a)

(A(B(H)(J))(C(D)(E(G))(F)))

(b)



(c)

Рис. 20. Способы изображения древовидных структур: (a) вложенные множества; (b) вложенные скобки; (c) список с отступами.

Алгебраическая формула неявным образом определяет древовидную структуру, которая часто обозначается другими средствами, причем либо вместе со скобками, либо совсем без них. Например, на рис. 21 показано дерево, соответствующее арифметическому выражению

$$a - b(c/d + e/f). \tag{2}$$

Согласно стандартным математическим соглашениям операции умножения и деления обладают приоритетом по сравнению с операциями сложения и вычитания. Благодаря этому приведенное выше выражение можно представить в упрощенном виде (2), а не в виде $a - \big(b \times \big((c/d) + (e/f)\big)\big)$, в котором все части выражения заключены в скобки. Эта связь между формулами и деревьями особенно важна при создании приложений.

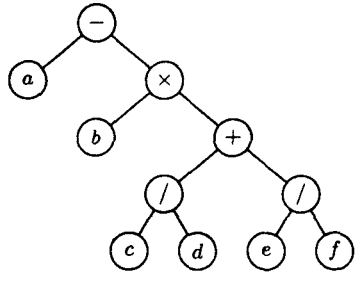


Рис. 21. Представление формулы (2) в виде дерева.

Обратите внимание на то, что список с отступами, показанный на рис. 20, (c), очень похож на оглавление данной книги. Действительно, даже сама эта книга обладает древовидной структурой. Например, древовидная структура главы 2 показана на рис. 22. Здесь следует отметить одну важную особенность: *нумерация разделов в настоящей книге представляет собой еще один способ представления древовидной структуры.* Этот метод часто называется десятичной системой обозначений Дьюи (Dewey decimal notation) по аналогии с подобной классификационной схемой, применяемой в библиотеках. Для дерева, представленного на рис. 19, она будет

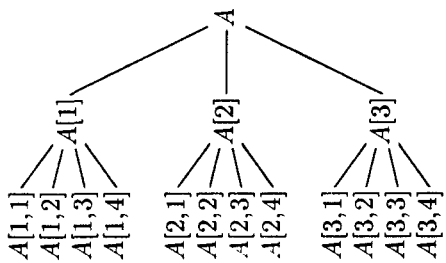
выглядеть так:

$$1\ A;\quad 1.1\ B;\quad 1.1.1\ H;\quad 1.1.2\ J;\quad 1.2\ C;$$
$$1.2.1\ D;\quad 1.2.2\ E;\quad 1.2.2.1\ G;\quad 1.2.3\ F.$$

Десятичную систему обозначений Дьюи можно применять по отношению к любому лесу: корень k-го дерева леса задается числом k, и, если α — количество узлов степени m, его дети будут обозначаться как $\alpha.1, \alpha.2, \ldots, \alpha.m$. Десятичная система обозначений Дьюи обладает многими простыми математическими свойствами и является полезным инструментом для анализа деревьев. Одним из примеров является естественное последовательное упорядочение узлов произвольного дерева, которое аналогично упорядочению разделов данной книги. Например, раздел 2.3 предшествует разделу 2.3.1 и располагается за разделом 2.2.6.

Между системой десятичных обозначений Дьюи и индексной системой обозначения переменных, которая неоднократно использовалась выше, существует довольно близкая связь. Если F — это лес деревьев, то $F[1]$ — все множество поддеревьев первого дерева, $F[1][2] \equiv F[1,2]$ — множество поддеревьев второго поддерева из множества $F[1]$, а $F[1,2,1]$ — первое множество поддеревьев первого поддерева из множества $F[1,2]$ и т. д. Обозначение узла $a.b.c.d$ в десятичной системе обозначений Дьюи соответствует узлу-родителю множества поддеревьев $(F[a,b,c,d])$. Это обозначение является расширением обычного индексного обозначения, поскольку допустимый диапазон значений каждого индекса зависит от значений индексов на предыдущих уровнях. Так, любой прямоугольный массив можно рассматривать как особый случай древовидной структуры, что и проиллюстрировано ниже на примере матрицы размера 3×4.

$$\begin{pmatrix} A[1,1] & A[1,2] & A[1,3] & A[1,4] \\ A[2,1] & A[2,2] & A[2,3] & A[2,4] \\ A[3,1] & A[3,2] & A[3,3] & A[3,4] \end{pmatrix}$$



Однако здесь следует отметить, что такая древовидная структура не совсем корректно отражает структуру матрицы, поскольку в ней представлены связи между элементами в пределах каждой строки, но отсутствуют связи между элементами в пределах каждого столбца.

В свою очередь, лес можно рассматривать как особую *структуру списка* (*list structure*). Слово "список" здесь применяется в очень специфическом смысле, и, чтобы подчеркнуть это, его пишут с прописной буквы: "Список". Рекурсивно Список определяется как *конечная последовательность атомов или Списков, число которых может быть больше или равно нулю.* Здесь под словом "атом" подразумевается неопределенное понятие, которое может относиться к элементам любой совокупности объектов и которое можно отличить от Списка. С помощью обычной системы обозначений на основе запятых и скобок можно различать атомы и Списки, а также быстро и просто указывать упорядочение в пределах Списка.

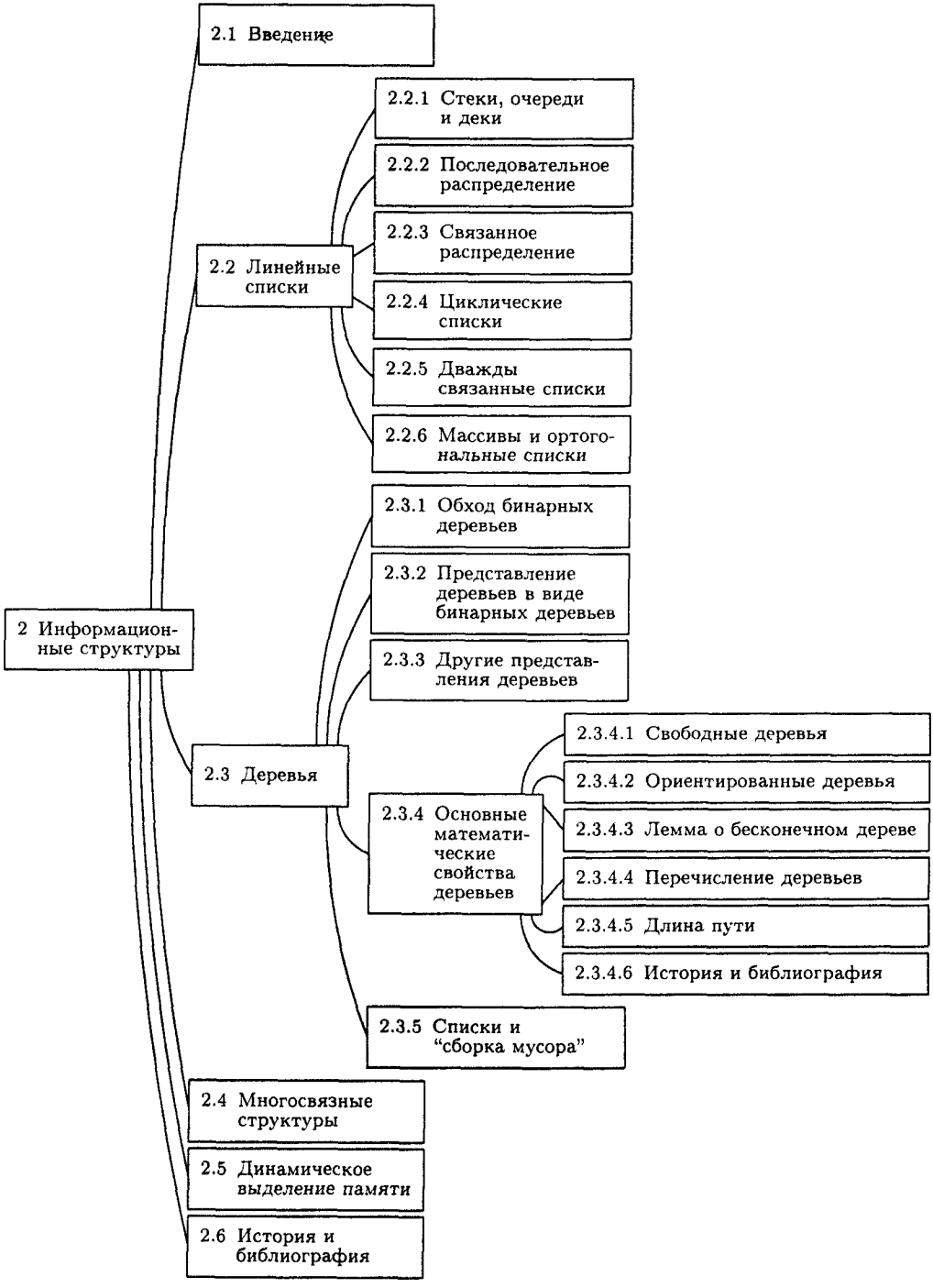


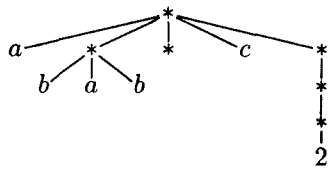
Рис. 22. Структура главы 2.

Рассмотрим, например, Список с пятью элементами

$$L = (a,\ (b, a, b),\ (\ ),\ c,\ (((2)))), \tag{3}$$

в котором сначала следует атом a, затем — Список (b, a, b), после — пустой Список $(\ )$, атом c и, наконец, Список $(((2)))$. Последний Список состоит из Списка $((2))$, который включает Список (2), который, в свою очередь, включает атом 2.

Этому Списку соответствует такая древовидная структура:



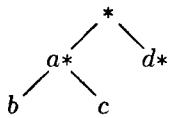
(4)

Звездочки используются для обозначения Списков, чтобы их можно было отличить от атомов. Индексные обозначения могут применяться для Списков точно так, как и для леса, например $L[2] = (b, a, b)$ и $L[2, 2] = a$.

Узлы-Списки на схеме (4) не несут никакой другой полезной информации, помимо того, что они являются Списками. Для устранения этого недостатка их можно пометить символами, как было сделано выше для деревьев и других структур. Так, обозначение

$$A = (a{:}(b, c),\ d{:}(\ ))$$

могло бы соответствовать дереву



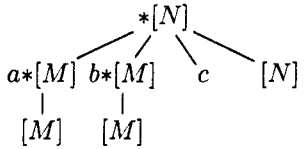
Важное отличие между Списками и деревьями заключается в том, что Списки могут перекрываться (т. е. подсписки могут пересекаться) и даже быть рекурсивными (т. е. содержать самих себя). Например, Список

$$M = (M), \tag{5}$$

как и Список

$$N = (a{:}M, b{:}M, c, N), \tag{6}$$

не соответствует никакой древовидной структуре. (В этих примерах прописными буквами указаны Списки, а строчными — ярлыки и атомы.) Структуры (5) и (6) с помощью звездочки, обозначающей Список, можно схематически отобразить таким образом:



(7)

На самом деле Списки не так уж сложно устроены, как может показаться после ознакомления с приведенными выше примерами. По сути, они являются простым

обобщением линейных списков, которые рассматривались в разделе 2.2, с дополнительным условием, что элементы линейных Списков могут быть переменными связи, которые указывают на другие линейные Списки (и, возможно, на самих себя).

Резюме. Четыре тесно связанных типа информационных структур — деревья, леса, бинарные деревья и Списки — имеют разное происхождение, поэтому они очень важны для компьютерных алгоритмов. В настоящей главе представлены различные способы схематического изображения этих структур, а также рассмотрены некоторые термины и понятия, используемые при работе с ними. В следующих разделах данные идеи рассматриваются более подробно.

УПРАЖНЕНИЯ

1. [*18*] Сколько различных деревьев можно создать на основе узлов A, B и C?

2. [*20*] Сколько разных *ориентированных* деревьев можно создать на основе узлов A, B и C?

3. [*M20*] Докажите, опираясь только на определения, что для каждого узла X дерева существует единственный путь к корню, а именно — единственная последовательность $k \geq 1$ *узлов* $X_1, X_2, \ldots, X_k$, *таких, что* X_1 *является корнем узла,* $X_k = X$, а X_j — родителем X_{j+1} для $1 \leq j < k$. (Этот метод типичен для доказательств почти всех элементарных фактов о древовидных структурах.) *Указание.* Примените метод индукции по отношению к количеству узлов дерева

4. [*01*] Верно ли следующее утверждение: "Если в обычной схеме дерева (с корнем сверху) узел X обладает *большим* номером уровня, чем узел Y, то узел X располагается в этой схеме *ниже* узла Y"?

5. [*02*] Если узел A имеет трех братьев-сестер, а узел B является родителем узла A, то чему равна степень узла B?

▶ **6.** [*21*] Дайте определение отношения "X является m-родным кузеном (1-родные кузены — это двоюродные братья, 2-родные кузены — это троюродные братья и т. д.) Y в n-м колене" (т. е X на n поколений старше Y) для узлов X и Y дерева по аналогии с генеалогическим деревом, если $m > 0$ и $n \geq 0$. (Значения этих терминов в контексте генеалогических деревьев найдите в словаре.)

7. [*23*] Расширьте определение из предыдущего упражнения для всех $m \geq -1$ и для всех целых чисел $n \geq -(m+1)$ таким образом, чтобы для любых двух узлов X и Y дерева существовали такие единственные m и n, что X является m-родным братом-сестрой узла Y в n-м колене.

▶ **8.** [*03*] Какое бинарное дерево не является деревом?

9. [*00*] Какой узел (B или A) в двух бинарных деревьях (1) является корнем?

10. [*M20*] Набор непустых множеств называется *вложенным*, если для заданной пары множеств X и Y либо $X \subseteq Y$, либо $X \supseteq Y$, либо X и Y не пересекаются. (Иначе говоря, *пересечением* $X \cap Y$ *является либо* X, *либо* Y, *либо* $\emptyset$.) На рис. 20, (а) показано, что любое дерево соответствует набору вложенных множеств. Верно ли обратное утверждение: каждый такой набор соответствует дереву?

▶ **11.** [*HM32*] Расширим определение дерева для включения понятия бесконечного дерева, рассмотрев наборы вложенных множеств из упр 10. Можно ли для каждого узла бесконечного дерева определить понятия "уровень", "степень", "родитель" и "ребенок"? Приведите примеры вложенных множеств действительных чисел, соответствующих дереву, в котором

а) каждый узел обладает несчетной степенью и имеется бесконечное множество уровней;

b) имеются узлы с несчетным уровнем,

c) каждый узел имеет по крайней мере степень 2 и существует несчетное множество уровней

12. [*M23*] При каких условиях частично упорядоченное множество соответствует неупорядоченному дереву или лесу? (Частично упорядоченные множества определены в разделе 2 2 3)

13. [*10*] Предположим, что номер узла X в десятичной системе обозначений Дьюи равен $a_1\ a_2\ \ \ a_k$ Какими тогда будут номера узлов на пути от X к корню (см упр 3) в этой системе обозначений?

14. [*M22*] Пусть S — это любое непустое множество элементов $1\ a_1\ \ \ a_k$, где $k \geq 0$ и $a_1, \ \ , a_k$ — положительные целые числа Покажите, что S соответствует дереву в том случае, когда оно конечно и удовлетворяет следующему условию если $\alpha\ m$ принадлежит этому множеству, то ему также принадлежит $\alpha\ (m-1)$, если $m > 1$, или α, если $m = 1$ (Это условие, очевидно, выполняется для дерева в десятичной системе обозначений Дьюи, следовательно, его можно рассматривать как еще один способ определения древовидной структуры)

▸ **15.** [*20*] Придумайте систему обозначений для узлов бинарного дерева, аналогичную десятичной системе обозначений Дьюи для узлов деревьев

16. [*20*] Нарисуйте схемы, аналогичные изображенным на рис 21 которые соответствуют следующим арифметическим выражениям (a) $2(a - b/c)$, (b) $a + b + 5c$

17. [*01*] Если представленную на рис 19 структуру рассматривать как лес F, то какому узлу будет соответствовать узел-родитель множества поддеревьев $(F[1\ 2, 2])$?

18. [*08*] *Каким элементам в Списке* (3) *соответствуют обозначения* $L[5\ 1, 1]$ *и* $L[3\ 1]$?

19. [*15*] Создайте схему Списка, аналогичную схеме (7), для Списка $L = (a, (L))$ Каким элементам этого Списка соответствуют обозначения $L[2]$ и $L[2\ 1, 1]$?

▸ **20.** [*M21*] Пусть *0-2-дерево* обозначает дерево в котором каждый узел не имеет ни одного потомка или имеет двух детей (Формально 0-2-дерево состоит из одного узла — корня, плюс 0 или 2 непересекающихся 0-2-деревьев) Покажите, что каждое 0-2-дерево имеет нечетное число узлов, и установите взаимно однозначное соответствие между бинарными деревьями с n узлами и (упорядоченными) 0-2-деревьями с $2n + 1$ узлами

21. [*M22*] Если дерево имеет n_1 узлов степени 1, n_2 узлов степени 2, и n_m узлов степени m, то сколько в нем содержится концевых узлов?

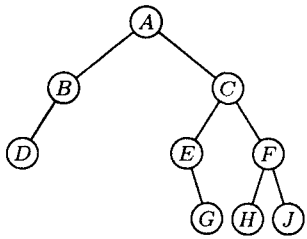
▸ **22.** [*21*] Используемые в Европе стандартные форматы страниц A0, A1, A2, , An, представляют собой прямоугольники с соотношением сторон $\sqrt{2}$ к 1 и площадью 2^{-n} квадратных метров Следовательно, при разрезании пополам страницы формата An получим две страницы формата A$(n + 1)$ Используя данный принцип, создайте графическое представление бинарных деревьев и проиллюстрируйте эту идею на примере структуры 2 3 1–(1) из следующего раздела

2.3.1. Обход бинарных деревьев

Прежде чем продолжить изучение деревьев, необходимо тщательно рассмотреть свойства бинарных деревьев, поскольку деревья общего типа обычно представляются внутри компьютера в виде некоторого эквивалентного бинарного дерева

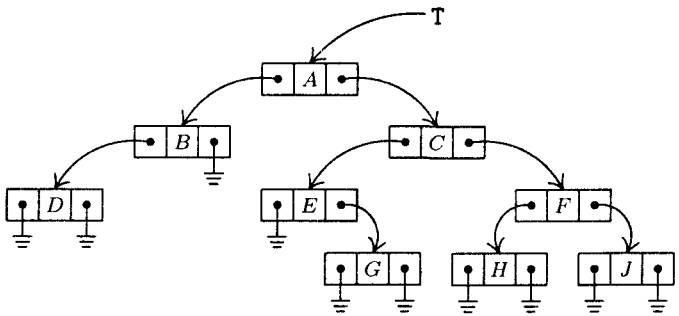
Бинарное дерево определено выше как конечное множество узлов, которое может либо быть пустым, либо состоять из корня вместе с двумя другими бинарными

деревьями. Это определение наводит на мысль о естественном способе представления бинарных деревьев внутри компьютера: каждый узел имеет связи LLINK и RLINK, а также переменную связи T, которая является указателем дерева. Если дерево пусто, то T = Λ; в противном случае T является адресом корневого узла этого дерева, а LLINK(T), RLINK(T) — указателями на левое и правое поддеревья этого корня соответственно. Данные правила рекурсивно определяют представление в памяти любого бинарного дерева. Например, дерево



(1)

может быть представлено таким образом:



(2)

Это простое и естественное представление дерева в памяти компьютера и объясняет особую важность бинарных древовидных структур Как показано в разделе 2.3.2, деревья общего типа очень удобно представлять в виде бинарных деревьев. Более того, многие деревья в приложениях сами по себе являются бинарными, поэтому они представляют особый интерес.

Существует достаточно много алгоритмов работы с древовидными структурами, в которых наиболее часто встречается понятие *обхода* (*traversing*) дерева или "прохода" по дереву. При таком методе исследования дерева каждый узел посещается в точности один раз, а полный обход дерева задает линейное упорядочение узлов, что позволяет упростить алгоритм, так как при этом можно использовать понятие "следующий" узел, т. е. узел, который располагается перед данным узлом в таком упорядочении или после него.

Для обхода бинарного дерева можно применить один из трех принципиально разных способов: *в прямом порядке* (*preorder*), *в центрированном порядке* (*inorder*) или *в обратном порядке* (*postorder*). Эти три метода определяются рекурсивно. Если бинарное дерево пусто, то для его "обхода" ничего делать не потребуется,

в противном случае обход выполняется в три этапа.

Прямой порядок обхода	Центрированный порядок обхода
Попасть в корень	Пройти левое поддерево
Пройти левое поддерево	Попасть в корень
Пройти правое поддерево	Пройти правое поддерево

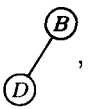
Обратный порядок обхода

Пройти левое поддерево
Пройти правое поддерево
Попасть в корень

Если применить эти определения к бинарному дереву (1) и (2), то при прямом порядке обхода узлов получим последовательность

$$A \quad B \quad D \quad C \quad E \quad G \quad F \quad H \quad J. \tag{3}$$

(Сначала следует корень A, затем — левое поддерево в прямом порядке,



,

и, наконец, правое поддерево в прямом порядке.) При центрированном порядке обхода корень посещается после обхода узлов одного из деревьев точно так, как если бы узлы дерева "проектировались" на горизонтальную прямую с образованием последовательности

$$D \quad B \quad A \quad E \quad G \quad C \quad H \quad F \quad J. \tag{4}$$

Аналогично обратный порядок обхода позволяет получить последовательность

$$D \quad B \quad G \quad E \quad H \quad J \quad F \quad C \quad A. \tag{5}$$

Как будет показано ниже, эти три способа упорядочения узлов бинарного дерева в виде линейной последовательности чрезвычайно важны, поскольку они тесно связаны с большинством компьютерных методов обработки деревьев. Названия *прямой*, *центрированный* и *обратный* происходят, конечно, от расположения корня по отношению к поддеревьям. Во многих приложениях бинарных деревьев понятия левого и правого поддеревьев совершенно симметричны, и в таких случаях термин *симметричный порядок* используется в качестве синонима понятия *центрированный порядок*. Центрированный порядок, при котором корень располагается посередине, образует симметрию относительно левой и правой сторон: при отражении бинарного дерева относительно вертикальной оси получается простое обращение симметричного порядка.

Для применения в компьютерных вычислениях предложенные выше рекурсивные определения трех основных способов обхода придется сформулировать несколько иначе. Наиболее общие методы такой переформулировки рассматриваются в главе 8. Обычно для этого используется вспомогательный стек, например так, как в показанном ниже алгоритме.

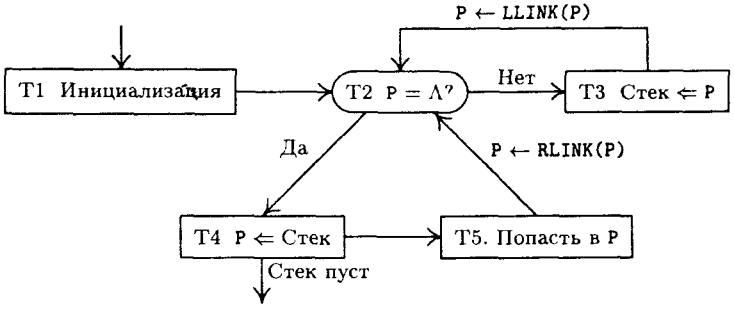


Рис. 23. Алгоритм T для симметричного обхода.

Алгоритм T (*Обход дерева в симметричном порядке*). Пусть T — указатель на бинарное дерево с представлением (2). Тогда в этом алгоритме (рис. 23) все узлы бинарного дерева обходятся в симметричном порядке с помощью вспомогательного стека A.

T1. [Инициализация.] Опустошить стек A и установить переменную связи P ← T.

T2. [P = Λ?] Если P = Λ, перейти к шагу T4.

T3. [Стек ⇐ P.] (Теперь P указывает на непустое бинарное дерево, которое нужно пройти.) Установить A ⇐ P, т. е. вставить (протолкнуть) значение P в стек A (см. раздел 2.2.1). Затем установить P ← LLINK(P) и вернуться к шагу T2.

T4. [P ⇐ Стек.] Если стек A пуст, выполнение алгоритма прекращается; в противном случае установить P ⇐ A.

T5. [Попасть в P.] Попасть в узел NODE(P). Затем установить P ← RLINK(P) и вернуться к шагу T2. ▮

На заключительном шаге этого алгоритма слово "попасть" означает, что по мере обхода дерева при попадании в узел выполняется заранее предусмотренная обработка узлов. Алгоритм T по отношению к другим действиям основной программы выполняется как сопрограмма, т. е. основная программа вызывает эту сопрограмму для перехода от одного узла к другому в заданном симметричном порядке. Конечно, так как эта сопрограмма вызывает основную процедуру только в одном месте, она не очень отличается от подпрограммы (см. раздел 1.4.2). В алгоритме T предполагается, что в результате выполнения внешних действий в данном дереве не удаляется ни узел NODE(P), ни любой другой его узел-предшественник.

Чтобы понять идею, лежащую в основе этого алгоритма, читатель может в качестве полезного упражнения применить алгоритм T к бинарному дереву (2). По достижении шага T3 следует приступить к обходу бинарного дерева с корнем, на который указывает P. При этом основная идея заключается в сохранении указателя P в стеке с последующим обходом левого поддерева. После выполнения этих действий необходимо вернуться к шагу T4 и найти прежнее значение P в стеке. После обхода корня NODE(P) на шаге T5 остается только совершить обход правого поддерева.

Алгоритм T типичен для многих других алгоритмов, которые будут рассмотрены ниже, поэтому имеет смысл привести формальное доказательство утверждений из предыдущего абзаца. *Докажем* с помощью метода индукции, что алгоритм T

позволяет совершить обход бинарного дерева с n узлами в симметричном порядке. Наша цель будет достигнута, если доказать справедливость несколько более общего утверждения.

> *Начиная с шага T2 с указателем* P *на бинарное дерево с* n *узлами и стеком* A, *содержащим элементы* A[1] ... A[m] *для некоторого* $m \geq 0$, *эта процедура на шагах T2–T5 совершит обход бинарного дерева в симметричном порядке, а затем вернется к шагу T4 с возвратом стека* A *в его исходное состояние* A[1] ... A[m].

Для $n = 0$ это утверждение очевидно выполняется, как следствие шага T2. Если $n > 0$, пусть P_0 является значением указателя P в начале шага T2. Так как $P_0 \neq \Lambda$, выполним шаг T3, что для стека A означает его изменение с приведением к новому состоянию с элементами A[1] ... A[m]P_0, а P равняется LLINK(P_0). Теперь левое поддерево имеет меньше n узлов, а потому по индукции выполним обход левого поддерева в симметричном порядке и перейдем к шагу T4 со стеком, содержание которого равно A[1] ... A[m] P_0. На шаге T4 стек возвращается в исходное состояние A[1] ... A[m], а P $\leftarrow P_0$. На шаге T5 выполняется обход узла NODE(P_0) и устанавливается значение P $\leftarrow$ RLINK(P_0). Теперь правое поддерево имеет менее n узлов и по индукции совершаем обход правого поддерева в симметричном порядке с переходом к шагу T4. Таким образом выполнен обход всего дерева в симметричном порядке согласно определению этого порядка. Доказательство завершено.

Практически идентичный алгоритм можно сформулировать для обхода бинарных деревьев в прямом порядке (см. упр. 12). Несколько сложнее выглядит алгоритм обхода в обратном порядке (см. упр. 13), а потому подобный порядок не имеет такого большого значения, как два других.

Для указания узлов-последователей и узлов-предшественников в алгоритмах обхода бинарных деревьев удобно применять следующие обозначения. Если P указывает на узел бинарного дерева, то

P* — адрес узла-последователя NODE(P) при обходе в прямом порядке;
P$ — адрес узла-последователя NODE(P) при обходе в симметричном порядке;
P♯ — адрес узла-последователя NODE(P) при обходе в обратном порядке;
*P — адрес узла-предшественника NODE(P) при обходе в прямом порядке;
$P — адрес узла-предшественника NODE(P) при обходе в симметричном порядке;
♯P — адрес узла-предшественника NODE(P) при обходе в обратном порядке. (6)

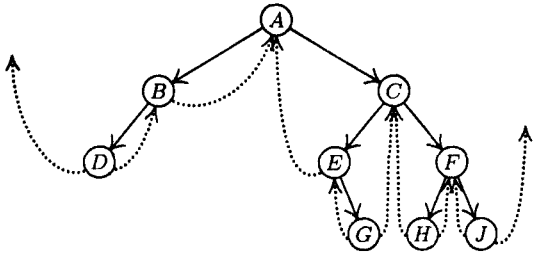
Если узел NODE(P) не имеет узлов-предшественников и узлов-последователей, то обычно используется обозначение LOC(T), где T — это внешний указатель на данное дерево. Таким образом, получаем $*(P*) = (*P)* = P$, $\$(P\$) = (\$P)\$ = P$ и $\sharp(P\sharp) = (\sharp P)\sharp = P$. В качестве примера использования этих обозначений предположим, что INFO(P) — это буква, изображенная в узле NODE(P) дерева (2). Тогда, если P указывает на его корень, получим INFO(P) = A, INFO(P*) = B, INFO(P$) = E, INFO($P) = B, INFO(♯P) = C и P♯ = *P = LOC(T).

Здесь у читателя может возникнуть чувство неуверенности в отношении правильности приведенных значений P*, P$ и т. д. Однако по мере изучения дальнейшего материала они станут более понятными, в частности для этого полезно выполнить упр. 16, приведенное в конце раздела. Символ "$" в обозначении "P$"

представляет букву S в английском написании термина "симметричный порядок" (symmetric order).

Существует еще один альтернативный вариант представления бинарных деревьев (2) в памяти компьютера, который отличается от предыдущего способа так же, как циклические списки отличаются от линейных однонаправленных списков. Обратите внимание на то, что в дереве (2) пустых связей содержится больше, чем всех остальных; и действительно, это верно для любого дерева, представленного с помощью обычного метода (см. упр. 14). На самом деле вряд ли стоит из-за этого так неэкономно расходовать пространство памяти. Вместо этого можно было бы, например, хранить в каждом узле некий двухбитовый "признак" (tag) того, что узел содержит либо пустую, либо непустую связь LLINK (или RLINK), либо обе пустые, либо обе непустые связи. В таком случае высвободившееся пространство в памяти, которое прежде использовалось для концевых связей, можно применять в других целях.

Хитроумный способ экономного использования памяти предложен А. Дж. Перлисом и Ч. Торнтоном , которые придумали метод *прошитого* (*threaded*) представления бинарного дерева. В этом методе концевые связи заменяются "нитями" (threads) которые связаны с другими частями дерева для упрощения его обхода. Прошитое дерево, которое эквивалентно дереву (2), выглядит так:



(7)

Здесь пунктиром обозначены "нити", которые всегда направлены к более высокому узлу дерева. *Каждый* узел теперь имеет две связи: одни узлы, например C, имеют две обычные связи с левым и правым поддеревьями, другие узлы, например H, — две связи в виде нитей, а третьи — по одной связи каждого типа. Особые нити, которые выходят из узлов D и J, будут рассмотрены ниже. Они появляются в крайнем слева и крайнем справа узлах.

Для представления прошитого бинарного дерева в памяти необходимо ввести обозначения, чтобы можно было отличить пунктирные и сплошные связи. Это может быть сделано, как предполагалось выше, с помощью двух дополнительных однобитовых полей в каждом узле, LTAG и RTAG. Тогда прошитое представление можно определить, например, следующим образом.

Обычное представление	Прошитое представление
LLINK(P) $= \Lambda$	LTAG(P) $= 1$, LLINK(P) $=$ \$P
LLINK(P) $=$ Q $\neq \Lambda$	LTAG(P) $= 0$, LLINK(P) $=$ Q
RLINK(P) $= \Lambda$	RTAG(P) $= 1$, RLINK(P) $=$ P\$
RLINK(P) $=$ Q $\neq \Lambda$	RTAG(P) $= 0$, RLINK(P) $=$ Q

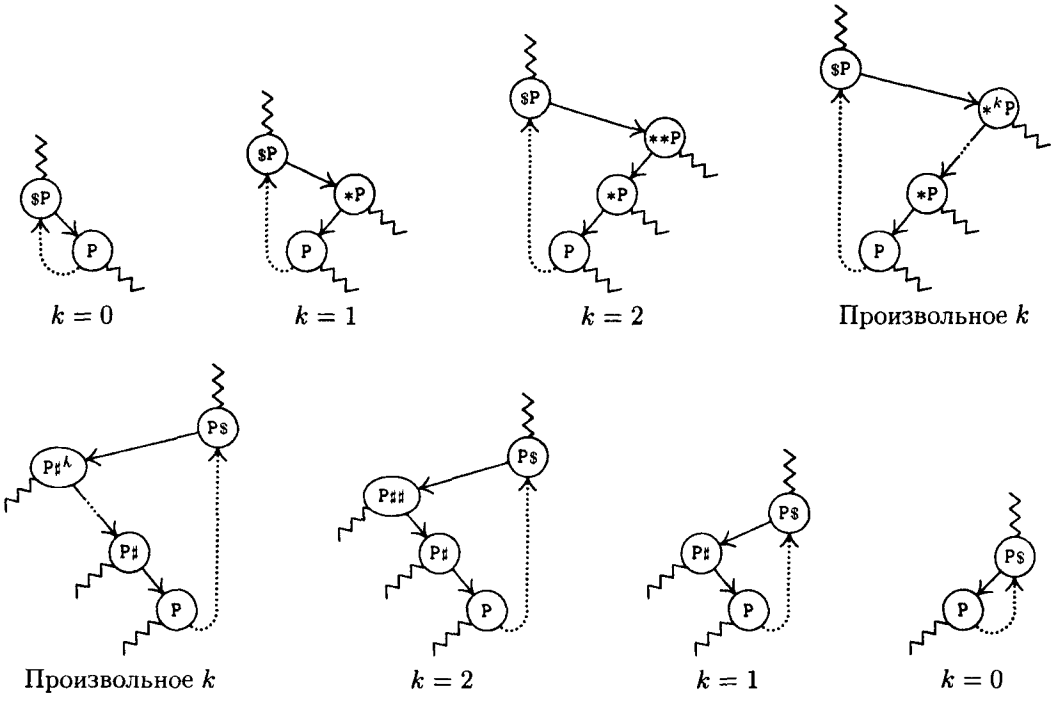


Рис. 24. Общая ориентация левых и правых связей-нитей в прошитом бинарном дереве. Волнистые линии обозначают связи с другими частями дерева (или ведущие к ним нити).

Согласно этому определению каждая новая связь-нить указывает непосредственно на узел-предшественник или узел-последователь конкретного узла в заданном симметричном порядке. На рис. 24 показана общая ориентация связей-нитей в любом бинарном дереве.

В некоторых алгоритмах гарантируется, что корень любого поддерева всегда располагается в ячейках памяти, которые находятся в памяти ниже других узлов этого поддерева. В таком случае `LTAG(P)` будет равно 1 тогда и только тогда, когда `LLINK(P) < P`, поэтому поле `LTAG`, как и поле `RTAG`, будет содержать избыточную информацию.

Значительным преимуществом прошитых деревьев является то, что алгоритмы обхода для них существенно упрощаются. Например, с помощью приведенного ниже алгоритма можно вычислить `P$` по заданному значению `P`.

Алгоритм S (*Симметричный (центрированный) узел-последователь в прошитом бинарном дереве*). Если `P` указывает на узел прошитого бинарного дерева, то данный алгоритм устанавливает `Q` $\leftarrow$ `P$`.

S1. [`RLINK(P)` — это нить?] Установить `Q` $\leftarrow$ `RLINK(P)`. Если `RTAG(P)` $= 1$, прекратить выполнение алгоритма.

S2. [Поиск слева.] Если `LTAG(Q)` $= 0$, установить `Q` $\leftarrow$ `LLINK(Q)` и повторить этот шаг. В противном случае прекратить выполнение алгоритма. ▮

Обратите внимание на то, что для его выполнения не потребуется стек, который применялся в алгоритме T. Действительно, с помощью обычного представления (2) невозможно найти P$ столь же эффективным путем, зная только адрес P произвольно выбранного узла дерева. Поскольку в обычном представлении бинарного дерева нет направленных вверх связей, то нет и сведений о том, какие узлы расположены выше, если только не сохранять информацию о пути к данному узлу. Стек в алгоритме T используется как раз для хранения этой информации при отсутствии нитей.

Алгоритм S назван эффективным, хотя это свойство не всегда очевидно, поскольку шаг S2 может выполняться достаточно много раз.. Может быть, вместо многократного повторения шага S2 для ускорения процесса следовало бы использовать стек, как в алгоритме T? Для исследования этого вопроса выясним, сколько раз в среднем выполнялся шаг S2, если P указывает "произвольный" узел дерева, или, что то же самое, определим общее количество выполнений шага S2, если алгоритм S повторно используется для обхода всего дерева.

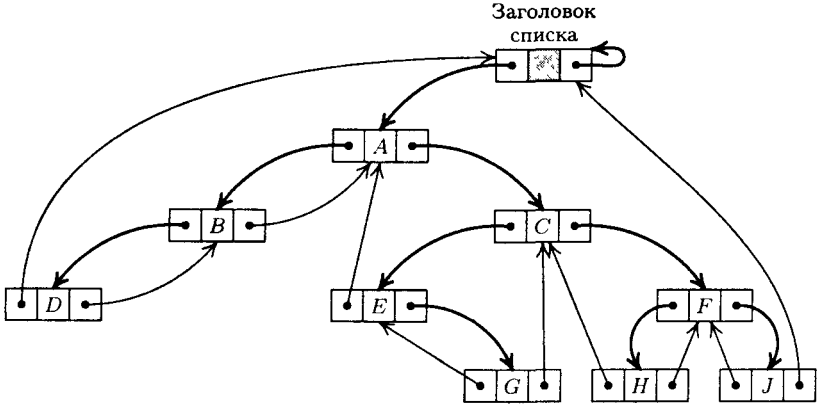
Выполняя этот анализ, полезно будет также познакомиться с программами, в которых реализованы алгоритмы S и T. Как обычно, при разработке алгоритмов следует учесть возможность их применения для пустых бинарных деревьев, и, если T является указателем данного дерева, желательно, чтобы LOC(T)* и LOC(T)$ были *первыми* узлами в прямом и симметричном порядках соответственно. Для прошитых деревьев узел NODE(LOC(T)) удобно преобразовать в "заголовок списка" для дерева со следующими параметрами:

$$\begin{aligned} &\texttt{LLINK(HEAD)} = \texttt{T}, \qquad &&\texttt{LTAG(HEAD)} = 0, \\ &\texttt{RLINK(HEAD)} = \texttt{HEAD}, \qquad &&\texttt{RTAG(HEAD)} = 0. \end{aligned} \tag{8}$$

(Здесь HEAD обозначает LOC(T), т. е. адрес заголовка списка.) Пустое прошитое дерево удовлетворяет условиям

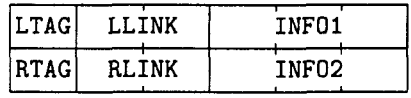
$$\texttt{LLINK(HEAD)} = \texttt{HEAD}, \quad \texttt{LTAG(HEAD)} = 1. \tag{9}$$

Дерево растет за счет вставки узлов *слева* от заголовка списка. (Эти начальные условия преимущественно продиктованы алгоритмом вычисления P*, который рассматривается в упр. 17.) В соответствии с этими соглашениями прошитое представление бинарного дерева (1) для компьютера будет выглядеть так:



(10)

После описания предварительных сведений можно приступать к созданию программ для компьютера MIX, предназначенных для реализации алгоритмов S и T. В приведенных ниже программах предполагается, что узлы бинарного дерева состоят из двух слов:



В непрошитом дереве LTAG и RTAG всегда будут равны "+", а концевые связи будут представлены нулем. В прошитом дереве знак "+" используется для меток, которые равны 0, а знак "−" — для меток, которые равны 1. Обозначения LLINKT и RLINKT будут использоваться для полей LTAG-LLINK и RTAG-RLINK соответственно.

Два бита метки занимают знаковые ячейки слова в памяти компьютера MIX, которые ни для чего другого все равно не используются, а потому они не занимают лишнего места в памяти. Аналогично в компьютере MMIX можно было бы "бесплатно" использовать *младшие* биты полей связи в качестве битов метки, поскольку указатель обычно принимает четные значения, а также потому что при адресации памяти в компьютере MMIX проще игнорировать именно младшие биты.

В следующих двух программах выполняется обход бинарного дерева в симметричном (т. е. центрированном) порядке с периодическими переходами к ячейке VISIT, в то время как индексный регистр 5 указывает на текущий узел.

Программа T. В этой реализации алгоритма T стек находится в ячейках A + 1, A + 2, ..., A + MAX; rI6 является указателем стека и rI5 ≡ P. Событие переполнения (OVERFLOW) происходит при недопустимом возрастании размеров стека. Эта программа незначительно отличается от алгоритма T (шаг T2 в нем встречается трижды), поэтому не нужно проверять, пуст ли стек, при переходе от шага T3 к шагу T2, а затем — к шагу T4.

```
01  LLINK EQU  1:2
02  RLINK EQU  1:2
03  T1    LD5  HEAD(LLINK)   1   T1. Инициализация. Установить P ← T.
04  T2A   J5Z  DONE          1   Стоп, если P = Λ.
05        ENT6 0              1
06  T3    DEC6 MAX            n   T3. Стек ⇐ P.
07        J6NN OVERFLOW       n   Емкость стека исчерпана?
08        INC6 MAX+1          n   Если нет, увеличить указатель стека.
09        ST5  A,6            n   Сохранить P в стеке.
10        LD5  0,5(LLINK)     n   P ← LLINK(P).
11  T2B   J5NZ T3             n   Перейти к шагу T3, если P ≠ Λ.
12  T4    LD5  A,6            n   T4. P ⇐ Стек.
13        DEC6 1              n   Уменьшить указатель стека.
14  T5    JMP  VISIT          n   T5. Попасть в P.
15        LD5  1,5(RLINK)     n   P ← RLINK(P).
16  T2C   J5NZ T3             n   T2. P = Λ?
17        J6NZ T4             a   Проверить, пуст ли стек.
18  DONE  ...                     ▮
```

Программа S. Алгоритм S дополнен условиями инициализации и прекращения выполнения, чтобы его программу можно было сравнить с программой T.

```
01  LLINKT EQU  0:2
02  RLINKT EQU  0:2
03  S0      ENT5 HEAD           1      S0. Инициализация. Установить P ← HEAD.
04          JMP  2F             1
05  S3      JMP  VISIT          n      S3. Попасть в P.
06  S1      LD5N 1,5(RLINKT)    n      S1. RLINK(P) — это нить?
07          J5NN 1F             n      Выполнить переход, если RTAG(P) = 1.
08          ENN6 0,5          n − a    В противном случае установить Q ← RLINK(P).
09  S2      ENT5 0,6            n      S2. Поиск слева. Установить P ← Q.
10  2H      LD6  0,5(LLINKT)  n + 1    Q ← LLINKT(P).
11          J6P  S2           n + 1    Если LTAG(P) = 0, повторить.
12  1H      ENT6 -HEAD,5      n + 1
13          J6NZ S3           n + 1    Переход к шагу S3 (необходимо попасть в P),
                                       если P ≠ HEAD. ∎
```

В приведенном выше коде показано также время выполнения отдельных команд, которое легко определить по закону Кирхгофа и следующим правилам.

i) В программе T количество вставок в стек должно равняться количеству удалений.

ii) В программе S поля LLINK и RLINK каждого узла проверяются только однажды.

iii) Количество "посещений" (visits) равно количеству узлов дерева.

Анализируя программу T, получим, что для ее выполнения потребуется $15n + a + 4$ единиц времени, а для программы S — $11n - a + 7$, где n — количество узлов в дереве, a — количество правых концевых связей (т. е. количество узлов без правого поддерева). Предполагая, что $n \neq 0$, получим, что число a может варьироваться от 1 до n. А если допустить существование симметрии между правой и левой сторонами дерева, то в таком случае среднее значение a будет равно $(n + 1)/2$. Доказательство данного результата приводится в упр. 14.

На основе этого анализа можно сделать следующие принципиальные выводы.

i) Если P — произвольно выбранный узел дерева, то шаг S2 в среднем выполняется только *однажды* за все время работы алгоритма S.

ii) Обход прошитого дерева происходит несколько быстрее, поскольку для него не нужно выполнять операции со стеком.

iii) Для алгоритма T требуется немного больше памяти, чем для алгоритма S, из-за использования вспомогательного стека. В программе T стек хранится в последовательных ячейках памяти, значит, на его размер следует наложить какое-то ограничение. При превышении этого ограничения могут возникнуть крайне нежелательные последствия, поэтому его следует выбрать достаточно большим (см. упр. 10). Поэтому требования к памяти со стороны программы T существенно выше, чем со стороны программы S. Часто в сложных программах нужно независимо выполнить обход сразу нескольких деревьев, и в каждом таком случае для работы программы T понадобится отдельный стек. Это предполагает использование в программе T связанного распределения для его стека (см. упр. 20). В такой ситуации время выполнения становится равным

$30n+a+4$ единицам, что приблизительно вдвое медленнее. Однако время обхода дерева может оказаться не таким уж и важным, если приходится учитывать еще и время выполнения другой сопрограммы. Еще один альтернативный вариант основан на хитроумном способе хранения внутри самого дерева связей стека (см. упр. 21).

iv) Алгоритм S, конечно, имеет более общий вид, чем алгоритм T, поскольку он позволяет сразу пройти от P к P$, когда нет необходимости совершать обход всего бинарного дерева.

Таким образом, прошитое бинарное дерево в отношении задачи обхода дерева обладает несомненными преимуществами по сравнению с непрошитым бинарным деревом. В некоторых приложениях эти преимущества практически сводятся на нет из-за несколько увеличенного времени выполнения вследствие вставки и удаления узлов в прошитом дереве. Иногда дополнительную экономию памяти можно получить за счет "совместного использования" общих поддеревьев с непрошитым представлением, тогда как для прошитых деревьев потребуется строго соблюсти древовидную структуру без какого-либо перекрытия поддеревьев.

Связи-нити также могут использоваться для вычисления P*, $P и ♯P, причем эффективность такого вычисления будет сравнима с эффективностью алгоритма S. Функции *P и P♯ несколько труднее вычислить, поскольку они предназначены для непрошитых представлений дерева. Читателю настоятельно рекомендуется выполнить упр. 17.

Преимущества прошитых деревьев могли бы быть утрачены в основном из-за сложности установления связей-нитей. Однако именно простота организации роста прошитых деревьев, которая реализуется почти так же легко. как и в случае роста обычных деревьев, определяет работоспособность данной идеи. В качестве примера рассмотрим следующий алгоритм.

Алгоритм I (*Вставка в прошитое бинарное дерево*). Этот алгоритм присоединяет один узел, NODE(Q), в качестве правого поддерева узла NODE(P), если правое поддерево пусто (т. е. если RTAG(P) = 1). В противном случае узел NODE(Q) вставляется между узлами NODE(P) и NODE(RLINK(P)) и последний из них становится правым ребенком узла NODE(Q). Предполагается, что бинарное дерево, в которое вставляется новый узел, является прошитым, как в (10). Один из вариантов этого алгоритма рассматривается в упр. 23.

I1. [Инициализация признаков и связей.] Установить RLINK(Q) ← RLINK(P), RTAG(Q) ← RTAG(P), RLINK(P) ← Q, RTAG(P) ← 0, LLINK(Q) ← P, LTAG(Q) ← 1.

I2. [Является ли RLINK(P) нитью?] Если RTAG(Q) = 0, установить LLINK(Q$) ← Q. (Здесь Q$ определяется алгоритмом S, который будет работать правильно, даже если LLINK(Q$) теперь указывает на узел NODE(P), а не на узел NODE(Q). Этот шаг необходим при вставке узла в середину прошитого дерева, а не при добавлении нового листа.) ▮

Поменяв местами левую и правую стороны (например, обозначение Q$ заменяя обозначением $Q на шаге I2), получим аналогичный алгоритм для вставки узла слева.

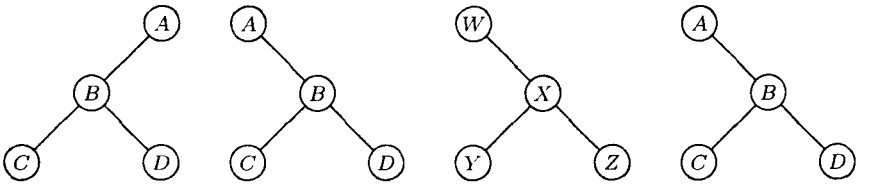
До сих пор рассматривались бинарные деревья, в которых связи-нити проводились слева и справа. Однако существует еще одно важное представление, которое

занимает промежуточное положение между непрошитым и полностью прошитым представлениями. Так называемое *правопрошитое бинарное дерево* (*right-threaded binary tree*) представляет собой комбинацию этих двух подходов за счет использования правых связей-нитей RLINK, тогда как пустые левые поддеревья представлены связями-нитями LLINK = Λ. (Аналогичным образом устроено левопрошитое бинарное дерево, но только в нем пустыми являются связи-нити LLINK.) В алгоритме S связи-нити LLINK практически не используются, поэтому, если заменить условие LTAG = 0 на шаге S2 условием LLINK $\neq \Lambda$, получим алгоритм обхода правопрошитого бинарного дерева в симметричном порядке. Причем программа S может без каких-либо изменений работать с правопрошитыми бинарными деревьями. Для очень многих приложений бинарных древовидных структур требуется выполнять обход дерева только слева направо с помощью функций P$ и/или P*, и потому для этих приложений нет необходимости прошивать связи-нити LLINK. Здесь рассмотрены случаи обхода в обоих направлениях (правом и левом) для того, чтобы указать на симметрию данной ситуации и ее возможности, но на практике прошивку гораздо чаще требуется выполнять только с одной стороны.

Рассмотрим теперь еще одно важное свойство бинарных деревьев и их связь с проблемой обхода узлов дерева. Говорят, что два бинарных дерева T и T' *подобны*, если они имеют одинаковую структуру. Формально это значит, что либо (a) они оба пусты, либо (b) они оба не пусты, а их левое и правое поддеревья подобны соответственно. Иначе говоря, подобие означает, что схемы деревьев T и T' имеют одинаковую "форму". Другими словами, подобие означает наличие взаимно однозначного соответствия между узлами деревьев T и T' с сохранением структуры. Если узлы u_1 и u_2 дерева T соответствуют узлам u_1' и u_2' дерева T', то узел u_1 находится в левом поддереве u_2 тогда и только тогда, когда узел u_1' находится в левом поддереве u_2'. Это утверждение верно и для правых поддеревьев.

Бинарные деревья T и T' называются *эквивалентными* (*equivalent*), если они подобны и соответствующие узлы содержат одинаковую информацию. Формально пусть info(u) обозначает информацию, которая содержится в узле u; в таком случае деревья эквивалентны тогда и только тогда, когда либо (a) они оба пусты, либо (b) они оба не пусты и info(корень(T)) = info(корень(T')), а их левые и правые поддеревья соответственно эквивалентны.

В качестве примера этих определений рассмотрим следующие четыре бинарных дерева:



Первые два из них не являются подобными. Второе, третье и четвертое — подобны, а второе и четвертое — эквивалентны.

В некоторых приложениях, использующих древовидные структуры, необходимо определить подобие или эквивалентность бинарных деревьев. Для этого полезно рассмотреть следующую теорему.

Теорема А. *Пусть*

$$u_1, u_2, \ldots, u_n \qquad \text{и} \qquad u'_1, u'_2, \ldots, u'_{n'}$$

являются узлами бинарных деревьев T и T' соответственно в прямом порядке обхода. Для любого узла u положим

$$\begin{aligned} l(u) &= 1, \text{ если } u \text{ имеет непустое левое поддерево,} \quad \text{иначе } l(u) = 0; \\ r(u) &= 1, \text{ если } u \text{ имеет непустое правое поддерево,} \quad \text{иначе } r(u) = 0; \end{aligned} \tag{11}$$

Следовательно, T и T' подобны тогда и только тогда, когда $n = n'$ и

$$l(u_j) = l(u'_j), \qquad r(u_j) = r(u'_j) \qquad \text{для } 1 \le j \le n. \tag{12}$$

Более того, деревья T и T' эквивалентны тогда и только тогда, когда

$$\text{info}(u_j) = \text{info}(u'_j) \qquad \text{для } 1 \le j \le n. \tag{13}$$

Обратите внимание, что l и r представляют собой дополнения к LTAG и RTAG в прошитом дереве. Эта теорема характеризует любую бинарную древовидную структуру на основании двух последовательностей нулей и единиц.

Доказательство. Ясно, что данное условие эквивалентности бинарных деревьев будет автоматически выполняться, если доказать заданное условие подобия. Более того, условия $n = n'$ и (12), конечно же, необходимы, так как соответствующие узлы подобных деревьев должны занимать одинаковые позиции при прямом порядке обхода. Следовательно, достаточно доказать, что деревья T и T' подобны, если выполняется условие (12) и $n = n'$. Доказательство осуществляется методом индукции по n с помощью следующего вспомогательного результата.

Лемма Р. *Пусть $u_1, u_2, \ldots, u_n$ являются узлами непустого бинарного дерева в прямом порядке обхода, а $f(u) = l(u) + r(u) - 1$. Тогда*

$$f(u_1) + f(u_2) + \cdots + f(u_n) = -1 \quad \text{и} \quad f(u_1) + \cdots + f(u_k) \ge 0, \quad 1 \le k < n. \tag{14}$$

Доказательство. Утверждение очевидно для $n = 1$. Если $n > 1$, бинарное дерево состоит из корня u_1 и других узлов. Если $f(u_1) = 0$, то либо левое поддерево, либо правое поддерево является пустым, поэтому данное условие, очевидно, истинно по индукции. Если $f(u_1) = 1$, предположим, что левое поддерево содержит n_l узлов; далее, используя метод индукции, получаем

$$f(u_1) + \cdots + f(u_k) > 0 \quad \text{для } 1 \le k \le n_l, \quad f(u_1) + \cdots + f(u_{n_l+1}) = 0, \tag{15}$$

и условие (14) снова становится очевидным. ▮

(С другими теоремами, аналогичными лемме Р, можно ознакомиться при обсуждении польской системы обозначений в главе 10.)

Для завершения доказательства теоремы А заметим, что она, очевидно, верна при $n = 0$. Если $n > 0$, то из определения прямого порядка следует, что u_1 и u'_1 являются соответствующими корнями этих деревьев и существуют такие n_l и n'_l (размеры левого поддерева), что

$u_2, \ldots, u_{n_l+1}$ и $u'_2, \ldots, u'_{n'_l+1}$ — левые поддеревья деревьев T и T';
$u_{n_l+2}, \ldots, u_n$ и $u'_{n'_l+2}, \ldots, u'_n$ — правые поддеревья деревьев T и T'.

Доказательство этой теоремы можно завершить с помощью метода индукции, если показать, что $n_l = n'_l$. При этом возможны три таких случая:

если $l(u_1) = 0$, то $n_l = 0 \neq n'_l$;
если $l(u_1) = 1$, $r(u_1) = 0$, то $n_l = n - 1 = n'_l$;
если $l(u_1) = r(u_1) = 1$, то по лемме P можно найти такое наименьшее $k > 0$, что $f(u_1) + \cdots + f(u_k) = 0$; и тогда $n_l = k - 1 = n'_l$ (см. (15)). ▮

Как следствие теоремы A, проверка эквивалентности или подобия двух прошитых бинарных деревьев может быть выполнена просто за счет прямого их обхода и проверки полей INFO и TAG. Некоторые интересные расширения теоремы A получены А. Я. Бликлом [A. J. Blikle, *Bull. de l'Acad. Polonaise des Sciences*, Série des Sciences Math., Astr., Phys., **14** (1966), 203–208]. Он рассмотрел бесконечный класс возможных порядков обхода, из которых только шесть (включая прямой порядок обхода) из-за их простых свойств были названы безадресными.

Завершим этот раздел описанием типичного алгоритма для бинарных деревьев, который копирует бинарное дерево в другие ячейки памяти.

Алгоритм C (*Копирование бинарного дерева*). Пусть HEAD — адрес заголовка списка бинарного дерева T; таким образом, T — левое поддерево HEAD, а LLINK(HEAD) — его адрес. Пусть NODE(U) — *узел с пустым левым поддеревом*. Этот алгоритм копирует T, и копия становится левым поддеревом узла NODE(U). В частности, если узел NODE(U) является заголовком списка пустого бинарного дерева, алгоритм превращает пустое дерево в копию дерева T.

C1. [Инициализация.] Установить P ← HEAD, Q ← U. Перейти к шагу C4.

C2. [Есть ли что-либо справа?] Если NODE(P) имеет непустое правое поддерево, установить R ⇐ AVAIL и присоединить NODE(R) справа к узлу NODE(Q). (В начале шага C2 правое поддерево узла NODE(Q) пусто.)

C3. [Копирование INFO.] Установить INFO(Q) ← INFO(P). (Здесь INFO обозначает все части узла, которые следует копировать, за исключением связей.)

C4. [Есть ли что-либо слева?] Если NODE(P) имеет непустое левое поддерево, установить R ⇐ AVAIL и присоединить NODE(R) слева к узлу NODE(Q). (В начале шага C2 левое поддерево узла NODE(Q) пусто.)

C5. [Продвижение вперед.] Установить P ← P*, Q ← Q*.

C6. [Проверка завершения.] Если P = HEAD (или, что эквивалентно, если Q = RLINK(U), при условии, что NODE(U) имеет непустое правое поддерево), выполнение алгоритма прекращается; в противном случае перейти к шагу C2. ▮

Этот простой алгоритм демонстрирует типичный пример использования метода обхода дерева. В данном виде он может применяться для прошитых, непрошитых или частично прошитых деревьев. Для выполнения шага C5 требуется вычислить прямых последователей P* и Q*. Для непрошитых деревьев это обычно выполняется с помощью вспомогательного стека. Доказательство корректности алгоритма C представлено в упр. 29, а программа для компьютера MIX, соответствующая этому алгоритму для правопрошитого бинарного дерева, приводится в упр. 2.3.2–13. Для прошитых деревьев "присоединение" на шагах C2 и C4 выполняется с помощью алгоритма I.

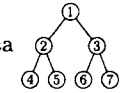
В приведенных ниже упражнениях содержится довольно много интересных задач, имеющих отношение к теме данного раздела.

Хотя бинарные или дихотомические системы, в принципе, регулярны, они являются едва ли не самыми неестественными типами упорядочения, которые только можно себе представить.

— ВИЛЬЯМ СВЭЙНСОН,
A Treatise on the Geography and Classification of Animals (1835)

УПРАЖНЕНИЯ

1. [*01*] Пусть INFO(P) в бинарном дереве (2) обозначает букву, которая хранится в узле NODE(P). Какой символ тогда хранится в узле INFO(LLINK(RLINK(RLINK(T))))?

2. [*11*] Перечислите узлы бинарного дерева  (a) в прямом порядке обхода; (b) в симметричном порядке обхода; (c) в обратном порядке обхода.

3. [*20*] Справедливо ли следующее утверждение: "Концевые узлы бинарного дерева встречаются в одном и том же относительном порядке для прямого, симметричного и обратного обходов"?

▸ **4.** [*20*] В данном разделе определяются три основных порядка обхода бинарного дерева. В дополнение к ним можно предложить еще один альтернативный способ обхода:

a) попасть в корень,
b) пройти правое поддерево,
c) пройти левое поддерево.

Далее это правило применяется рекурсивно для всех непустых поддеревьев. Имеет ли такой порядок какую-либо простую связь с тремя другими рассмотренными выше порядками?

5. [*22*] Узлы бинарного дерева можно идентифицировать последовательностью нулей и единиц, используя обозначения, подобные десятичной системе обозначений Дьюи для деревьев, следующим образом. Корень (если он существует) представляется последовательностью 1. Корни (если они существуют) левого и правого поддеревьев узла α соответственно представляются последовательностями $\alpha 0$ и $\alpha 1$. Например, узел H дерева (1) в данной системе обозначений выглядел бы как 1110 (см. упр. 2.3–15).

Покажите, что с помощью такой системы обозначений было бы удобно описывать прямой, симметричный и обратный порядки обхода.

6. [*M22*] Допустим, что бинарное дерево имеет n узлов, которые в прямом порядке обхода обозначаются как $u_1\, u_2 \ldots u_n$, а в симметричном порядке — как $u_{p_1}\, u_{p_2} \ldots u_{p_n}$. Покажите, что перестановку $p_1 p_2 \ldots p_n$ можно получить, передавая в стек последовательность $12 \ldots n$, как в упр. 2.2.1–2. И наоборот, покажите, что любая перестановка $p_1 p_2 \ldots p_n$, которую можно получить с помощью стека, соответствует некоторому бинарному дереву.

7. [*22*] Покажите, что, зная прямой и симметричный порядки узлов бинарного дерева, можно восстановить структуру бинарного дерева. Можно ли это сделать, зная прямой и обратный (вместо симметричного) порядки или симметричный и обратный порядки?

8. [*20*] Найдите все бинарные деревья, узлы которых располагаются в одинаковой последовательности (a) как в прямом, так и в симметричном порядках; (b) как в прямом, так и в обратном порядках; (c) как в симметричном, так и в обратном порядках.

9. [*M20*] Сколько раз выполняются шаги T1–T5 при обходе бинарного дерева с n узлами согласно алгоритму T в зависимости от n?

▸ **10.** [*20*] Какое максимальное количество элементов может находиться в стеке во время выполнения алгоритма T при работе с деревом с n узлами? (Ответ на этот вопрос очень важен при распределении памяти, когда стек располагается в последовательных ячейках памяти.)

11. [*HM41*] Найдите *среднее* значение для наибольшего размера стека при выполнении алгоритма T в зависимости от числа узлов n при условии, что все бинарные деревья с n узлами рассматриваются как равновероятные.

12. [*22*] Напишите алгоритм, аналогичный алгоритму T, который совершает обход бинарного дерева в *прямом порядке*, и докажите его корректность.

▸ **13.** [*24*] Напишите алгоритм, аналогичный алгоритму T, который совершает обход бинарного дерева в *обратном порядке*, и докажите его корректность.

14. [*22*] Покажите, что если бинарное дерево с n узлами имеет вид (2), то общее количество Λ-связей в таком представлении можно выразить с помощью простой функции от n. Эта функция не зависит от формы дерева

15. [*15*] В прошитом представлении дерева (10) каждый узел, за исключением заголовка списка, имеет в точности одну связь, которая указывает на нее сверху вниз, а именно — связь от родителя этого узла. Некоторые узлы также имеют связи, которые указывают на них снизу вверх. Например, узел C имеет два указателя, направленных на него снизу вверх, а узел E — только один. Существует ли простая зависимость между количеством связей, указывающих на данный узел, и другими основными свойствами узла? (Это нужно для того, чтобы при изменении структуры дерева знать количество связей, указывающих на узел.)

▸ **16.** [*22*] Схемы, представленные на рис 24, позволяют предложить следующие интуитивно понятные правила расположения узла `NODE(Q$)` в бинарном дереве по отношению к узлу `NODE(Q)`. Если узел `NODE(Q)` имеет непустое правое поддерево, рассмотрим `Q = $P`, `Q$ = P` в верхних схемах; узел `NODE(Q$)` является крайним слева узлом этого правого поддерева. Если узел `NODE(Q)` имеет пустое правое поддерево, рассмотрим `Q = P` в нижних схемах. Наконец, чтобы определить положение узла `NODE(Q$)`, следует перемещаться вверх по дереву до первой возможности перейти вправо

Предложите подобное "интуитивное" правило для поиска места расположения узла `NODE(Q*)` в бинарном дереве по отношению к узлу `NODE(Q)`.

▸ **17.** [*22*] Предложите алгоритм, аналогичный алгоритму S, для определения `P*` в прошитом бинарном дереве. Предполагается, что дерево имеет заголовок списка, как в (8)–(10).

18. [*24*] Во многих алгоритмах работы с деревьями каждый узел может посещаться не один раз, а *дважды* за счет комбинации прямого и симметричного порядка, который называется *двойным порядком*(*double order*). Обход бинарного дерева в двойном порядке определяется так: если бинарное дерево пусто, ничего делать не надо. В противном случае необходимо

a) посетить корень (первый раз);
b) пройти левое поддерево в двойном порядке;
c) посетить корень (во второй раз);
d) пройти правое поддерево в двойном порядке.

Например, после обхода дерева (1) в двойном порядке получим последовательность

$$A_1B_1D_1D_2B_2A_2C_1E_1E_2G_1G_2C_2F_1H_1H_2F_2J_1J_2,$$

где A_1 означает, что A посещается впервые.

Если `P` указывает на узел дерева и $d = 1$ или 2, положим $(\mathtt{P}, d)^{\triangle} = (\mathtt{Q}, e)$, если следующим шагом в двойном порядке после посещения узла `NODE(P)` в d-й раз является

посещение узла NODE(Q) в e-й раз; или, если $(\mathtt{P}, d)$ является последним шагом в двойном порядке, запишем $(\mathtt{P}, d)^{\Delta} = (\mathtt{HEAD}, 2)$, где HEAD — это адрес заголовка списка. Предположим также, что $(\mathtt{HEAD}, 1)^{\Delta}$ является первым шагом в двойном порядке.

Предложите алгоритм, аналогичный алгоритму S, для обхода бинарного дерева в двойном порядке, а также предложите алгоритм, аналогичный алгоритму S, для вычисления $(\mathtt{P}, d)^{\Delta}$. Рассмотрите связь между этими алгоритмами и алгоритмами из упр. 12 и 17.

▶ **19.** [*27*] Предложите алгоритм, аналогичный алгоритму S, для вычисления P♯ (a) в правопрошитом бинарном дереве; (b) в полностью прошитом бинарном дереве. Постарайтесь написать такой алгоритм, среднее время выполнения которого для произвольного узла дерева P было бы мало́, насколько возможно.

20. [*23*] Измените программу T так, чтобы стек использовался в ней в виде связанного списка, а не в виде последовательно расположенных ячеек памяти.

▶ **21.** [*33*] Создайте алгоритм обхода непрошитого бинарного дерева в симметричном порядке *без использования вспомогательного стека.* При этом допускается произвольное изменение содержимого полей LLINK и RLINK в узлах дерева во время его обхода при условии, что данное бинарное дерево имеет обычное представление (2) как до, так и после выполнения алгоритма обхода. Причем в узлах дерева нельзя использовать никакие другие биты.

22. [*25*] Создайте программу для компьютера MIX для реализации алгоритма из упр. 21 и сравните время ее выполнения со временем выполнения программ S и T.

23. [*22*] Предложите алгоритм, аналогичный алгоритму I, для вставки узла справа и слева в *правопрошитом* бинарном дереве. Предположим, что узлы содержат поля LLINK, RLINK и RTAG.

24. [*M20*] Будет ли справедлива теорема A, если узлы деревьев T и T' располагаются не в прямом, а в симметричном порядке?

25. [*M24*] Пусть $\mathcal{T}$ — множество бинарных деревьев, S — множество полей info, которые содержатся в узлах этих деревьев, причем S является линейно упорядоченным на основе отношения $\preceq$ (см. упр. 2.2.3-14). Определим отношение $T \preceq T'$ для любых деревьев T и T' из множества $\mathcal{T}$ тогда и только тогда, когда

i) T пусто; либо
ii) T и T' не пусты и $\mathrm{info}(\mathrm{root}(T)) \prec \mathrm{info}(\mathrm{root}(T'))$; либо
iii) T и T' не пусты, $\mathrm{info}(\mathrm{root}(T)) = \mathrm{info}(\mathrm{root}(T'))$, $\mathrm{left}(T) \preceq \mathrm{left}(T')$ и $\mathrm{left}(T)$ не эквивалентно $\mathrm{left}(T')$; либо
iv) T и T' не пусты, $\mathrm{info}(\mathrm{root}(T)) = \mathrm{info}(\mathrm{root}(T'))$, $\mathrm{left}(T)$ эквивалентно $\mathrm{left}(T')$ и $\mathrm{right}(T) \preceq \mathrm{right}(T')$.

Здесь $\mathrm{left}(T)$ и $\mathrm{right}(T)$ обозначают соответственно левое и правое поддеревья дерева T, а $\mathrm{root}(T)$ — корень дерева T. Докажите, что (a) из $T \preceq T'$ и $T' \preceq T''$ следует $T \preceq T''$; (b) T эквивалентно T' тогда и только тогда, когда $T \preceq T'$ и $T' \preceq T$; (c) для любого T, T' из множества $\mathcal{T}$ имеет место либо $T \preceq T'$, либо $T' \preceq T$. [Таким образом, если эквивалентные деревья множества $\mathcal{T}$ рассматриваются как равные, то отношение $\preceq$ порождает линейное упорядочение множества $\mathcal{T}$. Это упорядочение имеет много приложений (например, для упрощения алгебраических выражений). Если S содержит только один элемент, т. е. поля info всех элементов одинаковы, имеет место особый случай, когда эквивалентность равносильна подобию.]

26. [*M24*] Рассмотрим упорядочение $T \preceq T'$, определенное в предыдущем упражнении. Докажите теорему, аналогичную теореме A, которая дает необходимое и достаточное условие, что $T \preceq T'$, а также использует понятие двойного порядка обхода, которое определено в упр. 18.

▸ **27.** [*28*] Предложите алгоритм для проверки отношения двух деревьев T и T' (либо $T \prec T'$, либо $T \succ T'$, либо T эквивалентно T') на основании отношения, определенного в упр. 25, предполагая, что оба бинарных дерева являются правопрошитыми. Предположим, что каждый узел имеет поля LLINK, RLINK, RTAG, INFO, а вспомогательный стек не используется.

28. [*00*] Копия бинарного дерева, полученного с помощью алгоритма C, будет *эквивалентна* исходному дереву или *подобна* ему?

29. [*M25*] Предложите наиболее строгое доказательство справедливости алгоритма C.

▸ **30.** [*22*] Предложите алгоритм прошивки непрошитого дерева, который, например, преобразует (2) в (10). *Замечание.* По возможности используйте обозначения типа P* и P$ вместо повторения шагов алгоритмов обхода наподобие алгоритма T.

31. [*23*] Предложите алгоритм, который "стирает" правопрошитое бинарное дерево. Он должен возвратить все узлы дерева, за исключением заголовка списка, в список свободных ячеек AVAIL, причем заголовок списка отвечает пустому бинарному дереву. Предположим, что узел имеет поля LLINK, RLINK, RTAG, а вспомогательный стек не используется.

32. [*21*] Предположим, что каждый узел бинарного дерева имеет четыре поля связи: LLINK и RLINK, которые указывают на левое и правое поддеревья или Λ, как и в непрошитом дереве, а также SUC и PRED, которые указывают на предшественника и последователя узла в симметричном порядке. (Значит, SUC(P) = P$ и PRED(P) = $P. Такое дерево содержит больше информации, чем прошитое дерево.) Создайте алгоритм, подобный алгоритму I, для вставки в такое дерево.

▸ **33.** [*30*] Существует более одного способа прошивки дерева! Рассмотрим следующее представление, используя три поля LTAG, LLINK, RLINK в каждом узле:

LTAG(P): определяется так же, как и в прошитом бинарном дереве;
LLINK(P): всегда равно P*;
RLINK(P): определяется так же, как и в непрошитом бинарном дереве.

Обсудите алгоритмы вставки для такого представления и предложите для данного представления алгоритм копирования (т. е. алгоритм C) с подробным описанием.

34. [*22*] Пусть P указывает на узел в некотором бинарном дереве, а HEAD — на заголовок списка в пустом бинарном дереве. Предложите алгоритм, который (i) удаляет узел NODE(P) и все его поддеревья из любого дерева, а также (ii) присоединяет узел NODE(P) и его поддерево к NODE(HEAD). Предположим, что все рассматриваемые бинарные деревья прошиты справа с полями LLINK, RTAG, RLINK в каждом узле.

35. [*40*] Дайте определение *тройного дерева*(*ternary tree*) (а в более общем случае — t-арного дерева для произвольного $t \geq 2$), аналогичное определению бинарного дерева, и исследуйте возможность обобщения для t-арных деревьев результатов, полученных в этом разделе и в упражнениях после него.

36. [*M23*] В упр. 1.2.1–15 показано, что лексикографический порядок расширяет полное упорядочение множества S до полного упорядочения множеств из n элементов множества S. В приведенном выше упр. 25 показано, что линейное упорядочение данных в узлах дерева может быть расширено до линейного упорядочения деревьев на основе аналогичного определения. Если отношение $\prec$ приводит к полному упорядочению множества S, то является ли расширенное отношение из упр. 25 полным упорядочением набора $\mathcal{T}$?

▸ **37.** [*24*] (Задача Д. Фергюсона.) Если два слова в памяти компьютера обязательно должны содержать два поля связи и поле INFO с данными, то в таком случае для представления дерева типа (2) с n узлами потребуется $2n$ слов. Создайте схему представления бинарных

деревьев, для которой в памяти компьютера потребовалось бы выделить меньшее количество слов, предполагая, что *одна* связь и одно поле INFO с данными могут уместиться в одном слове.

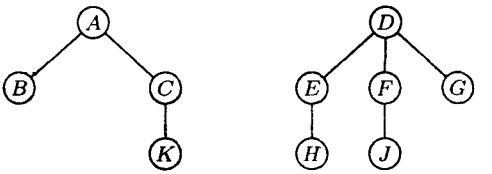
2.3.2. Представление деревьев в виде бинарных деревьев

Перейдем от бинарных деревьев к деревьям общего вида. Напомним следующие основные различия между деревьями и бинарными деревьями.

1) Дерево всегда имеет корень, т. е. оно никогда не бывает пустым; каждый узел может иметь 0, 1, 2, 3, ... детей.
2) Бинарное дерево может быть пустым, а каждый его узел может иметь 0, 1 или 2 детей; мы будем различать левых и правых детей.

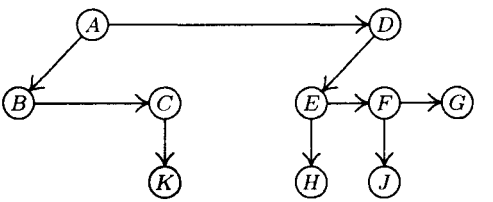
Напомним также, что лес является упорядоченным набором некоторого количества деревьев (и даже равного нулю). Поддеревья любого узла дерева образуют лес.

Существует естественный способ представления любого леса в виде бинарного дерева. Рассмотрим следующий лес, состоящий из двух деревьев:



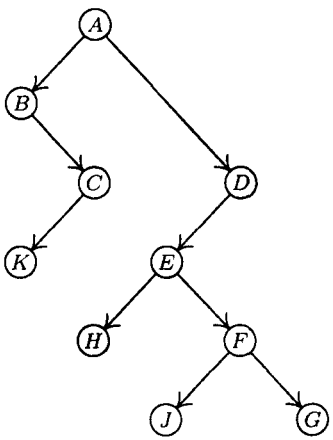
. (1)

Соответствующее бинарное дерево получим за счет связывания детей каждой семьи и удаления всех вертикальных связей, за исключением связи с родителем первого ребенка.



. **(2)**

Затем, наклонив эту схему под углом 45° и слегка откорректировав расположение узлов, получим такое бинарное дерево:



. (3)

И наоборот, после выполнения этих действий в обратном порядке становится очевидно, что любое бинарное дерево соответствует единственному лесу деревьев.

Приведение схемы (1) к схеме (3) имеет чрезвычайно большое значение. Оно называется *естественным соответствием* (*natural correspondence*) между лесом и бинарными деревьями. В частности, оно задает соответствие между деревьями и особым классом бинарных деревьев, а именно — между бинарными деревьями с корнем, но без правого поддерева. (При этом можно было бы слегка изменить точку зрения и допустить, что корень дерева соответствует заголовку списка бинарного дерева. В таком случае получим взаимно однозначное соответствие между деревьями с $n+1$ узлами и бинарными деревьями с n узлами.)

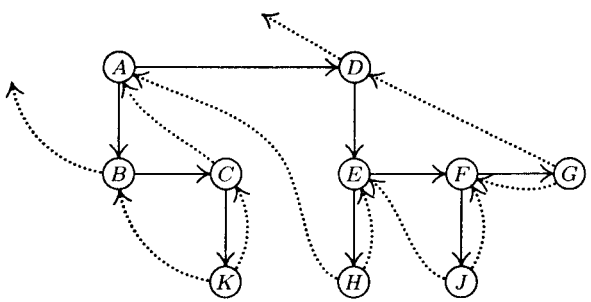
Пусть $F = (T_1, T_2, \ldots, T_n)$ — некоторый лес деревьев. Тогда бинарное дерево $B(F)$, соответствующее F, можно строго определить следующим образом.

a) Если $n = 0$, то $B(F)$ пусто.

b) Если $n > 0$, то корень $B(F)$ является корнем (T_1); $B(T_{11}, T_{12}, \ldots, T_{1m})$ является левым поддеревом дерева $B(F)$, где $T_{11}, T_{12}, \ldots, T_{1m}$ — поддеревья корня (T_1); $B(T_2, \ldots, T_n)$ является правым поддеревом дерева $B(F)$.

Эти правила строго определяют приведение схемы (1) к схеме (3).

Иногда удобно изображать схему бинарного дерева в виде (2), без поворота на 45°. Тогда соответствующее виду (1) *прошитое* (*threaded*) бинарное дерево будет выглядеть так:



(4)

(Ср. с рис. 24, повернув его на 45°.) Обратите внимание на то, что *правые связи-нити проходят от крайнего справа ребенка семьи к его родителю.* Левые связи-нити не имеют такой естественной интерпретации из-за отсутствия симметрии между левой и правой сторонами.

Представленные в предыдущем разделе идеи обхода можно теперь применить к лесу (т. е. к деревьям). Хотя простой аналогии симметричного порядка обхода здесь нет из-за отсутствия очевидного места вставки корня среди его наследников, зато прямой и обратный порядки можно перенести самым очевидным образом. Для заданного непустого леса эти два основных способа обхода можно определить, как показано ниже.

Обход в прямом порядке	Обход в обратном порядке
Посетить корень первого дерева	Пройти поддеревья первого дерева
Пройти поддеревья первого дерева	Посетить корень первого дерева
Пройти оставшиеся деревья	Пройти оставшиеся деревья

Чтобы понять значение этих двух методов обхода, рассмотрим следующее обозначение древовидной структуры на основе вложенных скобок:

$$(A(B, C(K)),\ D(E(H),\ F(J),\ G)). \tag{5}$$

Оно соответствует лесу (1). При этом сначала дерево представляется символом из его корня, а затем дается представление его поддеревьев. В результате представление непустого леса имеет вид заключенного в скобки списка представлений его деревьев, которые разделены запятыми.

Если обход осуществляется (1) в прямом порядке, то узлы посещаются в последовательности $A\,B\,C\,K\,D\,E\,H\,F\,J\,G$, которая идентична обозначению (5), но без скобок и запятых. Прямой порядок является естественным способом перечисления узлов дерева: сначала указывается корень, а затем — его потомки. Если древовидная структура представлена с помощью строк с отступами, как на рис. 20, (*c*), то строки располагаются в прямом порядке. В данной книге разделы нумеруются в прямом порядке (см. рис. 22). Таким образом, например, за разделом 2.3 следует раздел 2.3.1, а затем — разделы 2.3.2, 2.3.3, 2.3.4, 2.3.4.1, ..., 2.3.4.6, 2.3.5, 2.4 и т. д.

Интересно отметить, что прямой порядок является освященным веками понятием *династический порядок* (*dynastic order*). После смерти короля, герцога или графа соответствующий титул передается первому (т. е. старшему) сыну, затем — наследникам первого сына и, если таковых нет, другим сыновьям семьи в том же порядке. (В Англии титул может точно так перейти и к дочери, но это возможно только после смерти (или при отсутствии) всех сыновей.) Теоретически родовой порядок всех узлов аристократических фамилий можно было бы переписать в прямом порядке. Тогда, рассматривая только живых представителей этих фамилий, можно получить *порядок престолонаследования* (за исключением тех, кто лишен этого права по закону об отречении от престола).

В обратном порядке узлы (1) имеют последовательность $B\,K\,C\,A\,H\,E\,J\,F\,G\,D$. Она аналогична прямому порядку, но при использовании обозначения на основе скобок,

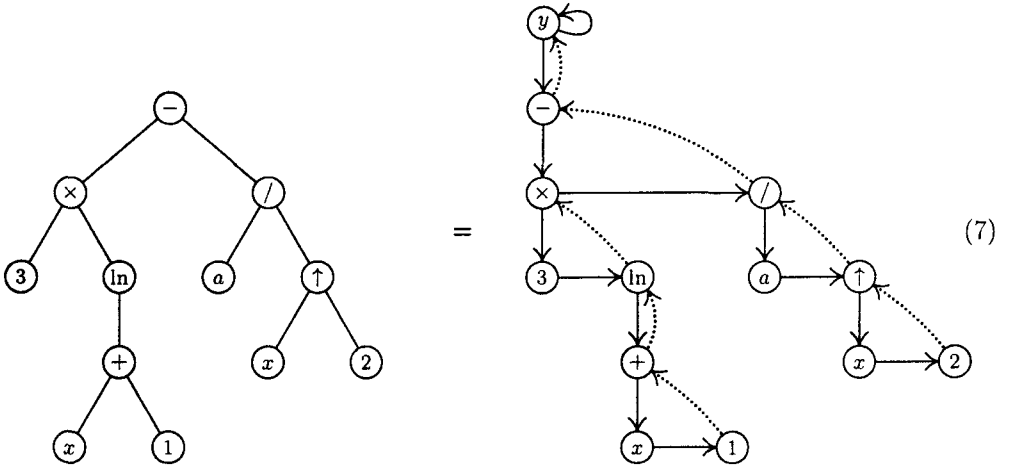
$$((B,\ (K)C)A,\ ((H)E,\ (J)F,\ G)D), \tag{6}$$

каждый узел располагается *после* своих наследников, а не перед ними.

Определения прямого и обратного порядков обхода прекрасно согласуются с естественным соответствием деревьев и бинарных деревьев. поскольку поддеревья первого дерева отвечают левому бинарному поддереву, а оставшиеся деревья — правому бинарному поддереву. Сравнив эти определения с соответствующими определениями на с. 364, получим, что обход леса в прямом порядке выполняется *точно так*, как обход соответствующего бинарного дерева в прямом порядке. Обход дерева в обратном порядке выполняется так же, как обход соответствующего бинарного дерева в *симметричном* порядке. Алгоритмы, рассмотренные в разделе 2.3.1, следовательно, могут быть использованы без изменений. (Обратите внимание, что обратный порядок обхода деревьев соответствует симметричному, а *не* обратному порядку обхода бинарных деревьев. И это в определенной степени можно считать везением, поскольку, как было показано выше, довольно трудно выполнить обход бинарных деревьев в обратном порядке.) Вследствие этой эквивалентности впоследствии мы будем использовать обозначение `P$` для последователя узла `P` при

обратном порядке обхода дерева; в то же время этот символ обозначает последователя узла P при *симметричном* порядке обхода бинарного дерева.

В качестве примера использования этих методов для решения практических задач рассмотрим некоторые операции с алгебраическими формулами. Такие формулы лучше всего представлять в виде древовидных структур, а не как одно- или двумерные конфигурации символов, и даже не как бинарные деревья. Например, формулу $y = 3\ln(x+1) - a/x^2$ можно представить в виде дерева таким образом.



(7)

В левой части рисунка показано обычное дерево, подобное представленному на рис. 21, в котором бинарные операторы +, −, ×, / и ↑ (последний символ обозначает возведение в степень) имеют по два поддерева, соответствующих их операндам. Унарный оператор ln имеет одно поддерево, а переменные и константы являются концевыми узлами. В правой части рисунка показано эквивалентное правопрошитое бинарное дерево, включающее дополнительный узел y, который является заголовком списка этого дерева. Заголовок списка имеет вид, представленный условиями 2.3.1–(8).

Важно отметить, что, хотя дерево. показанное в левой части (7), внешне очень похоже на бинарное дерево, здесь оно рассматривается как *обычное дерево* и представлено в виде совершенно иного бинарного дерева, чем то, которое показано в правой части (7). Программы для выполнения алгебраических операций можно было бы создать непосредственно на основе бинарных деревьев, т. е. на основе так называемого трехадресного кода (three-address code) представления алгебраических формул. Однако на практике при использовании древовидного представления алгебраических формул применяются некоторые упрощения, подобно представлениям (7), поскольку обход дерева в обратном порядке выполняется проще в обычном дереве.

При обходе узлов дерева в левой части (7) получим

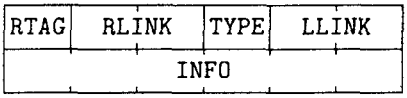
$$- \quad \times \quad 3 \quad \ln \quad + \quad x \quad 1 \quad / \quad a \quad \uparrow \quad x \quad 2 \qquad \text{для прямого порядка;} \qquad (8)$$

$$3 \quad x \quad 1 \quad + \quad \ln \quad \times \quad a \quad x \quad 2 \quad \uparrow \quad / \quad - \qquad \text{для обратного порядка.} \qquad (9)$$

Алгебраические выражения наподобие (8) и (9) имеют очень большое значение и называются польской системой обозначений, так как в виде (8) она впервые была

использована польским логиком Яном Лукасевичем (Jan Łukasiewicz). Выражение (8) является *префиксным обозначением* (*prefix notation*) формулы (7), а выражение (9) — ее *постфиксным обозначением* (*postfix notation*). В следующих главах эта система обозначений будет рассмотрена подробнее, а здесь лишь отметим, что польская система обозначений непосредственно связана с основными способами обхода деревьев.

Предположим, что узлы в древовидных структурах, используемых для представления алгебраических формул, имеют следующий вид в программах для компьютера MIX:



(10)

Здесь поля RLINK и LLINK имеют обычное значение, а значение поля RTAG является отрицательным для связей-нитей (что соответствует RTAG = 1 в выражениях алгоритма). Поле TYPE используется для того, чтобы можно было различать узлы разных типов. Например, TYPE = 0 означает, что узел представляет константу, а поле INFO содержит ее значение; TYPE = 1 означает, что узел представляет переменную, а поле INFO содержит ее пятибуквенное имя; TYPE $\geq$ 2 означает, что узел представляет оператор, а поле INFO содержит символьное имя этого оператора, т. е. значения TYPE = 2, 3, 4, ... используются для обозначения операторов $+$, $-$, $\times$, $/$ и т. д. Здесь нас меньше всего будет интересовать вопрос, как древовидная структура представлена в памяти компьютера, поскольку эта тема очень подробно анализируется в главе 10. Предположим лишь, что дерево уже размещено в памяти компьютера, и отложим на потом все вопросы, связанные с операциями ввода и вывода.

Рассмотрим классический пример выполнения алгебраических преобразований, а именно — поиск *производной* некоторого выражения как функции от x. Программы алгебраического дифференцирования, которые появились еще в 1952 году, были в числе первых программ для символьных вычислений.

На примере процесса дифференцирования можно проиллюстрировать многие методы алгебраических преобразований. к тому же он имеет большое практическое значение для выполнения теоретических расчетов в различных областях науки и техники.

Читатели, которые не знакомы с основами вычислительной математики, могут рассматривать эту задачу как абстрактное упражнение в области алгебраических преобразований, которые определяются следующими правилами.

$$D(x) = 1 \tag{11}$$

$$D(a) = 0, \quad \text{где } a \text{ — константа или переменная} \neq x \tag{12}$$

$$D(\ln u) = D(u)/u, \quad \text{где } u \text{ — любая формула} \tag{13}$$

$$D(-u) = -D(u) \tag{14}$$

$$D(u+v) = D(u) + D(v) \tag{15}$$

$$D(u-v) = D(u) - D(v) \tag{16}$$

$$D(u \times v) = D(u) \times v + u \times D(v) \tag{17}$$

$$D(u \,/\, v) \;= D(u)/v - \big(u \times D(v)\big)/(v \uparrow 2) \tag{18}$$

$$D(u \uparrow v) \;= D(u) \times \big(v \times (u \uparrow (v - 1))\big) + \big((\ln u) \times D(v)\big) \times (u \uparrow v) \tag{19}$$

Эти правила позволяют оценить производную $D(y)$ для любой формулы y, состоящей из перечисленных выше операторов. Знак "$-$" в правиле (14) является унарным оператором, который отличается от бинарного оператора "$-$" в (16). Далее для обозначения унарного отрицания в узлах деревьев будет использоваться обозначение "neg".

К сожалению, правил (11)–(19) недостаточно, так как, применив их вслепую для очень простой формулы

$$y = 3\ln(x + 1) - a/x^2,$$

получим правильный, но совершенно неудобный для дальнейшей работы результат:

$$\begin{aligned} D(y) = {} & 0 \cdot \ln(x + 1) + 3\big((1 + 0)/(x + 1)\big) \\ & - \big(0/x^2 - \big(a(1(2x^{2-1}) + ((\ln x) \cdot 0)x^2)\big)/(x^2)^2\big). \end{aligned} \tag{20}$$

Для сокращения количества лишних операций в этом результате нужно предусмотреть особые случаи сложения с нулем, умножения на нуль, умножения на единицу и возведения в степень "единица". Таким образом можно сократить выражение (20) с приведением его к виду

$$D(y) = 3\big(1/(x + 1)\big) - \big((-(a(2x)))/(x^2)^2\big). \tag{21}$$

Теперь оно стало более наглядным, но еще не совсем идеальным. На самом деле понятие удовлетворительного вида результата математических преобразований не очень хорошо определено, поскольку разные математики предпочитают разные способы представления итоговых формул. Однако ясно, что формула (21) не так проста, как могла бы быть. Чтобы представить ее в более приемлемом виде, необходимо разработать специальные программы упрощения алгебраических выражений (см. упр. 17), которые позволят привести формулу (21), например, к виду

$$D(y) = 3(x + 1)^{-1} + 2ax^{-3}. \tag{22}$$

Здесь мы ограничимся описанием программ, с помощью которых можно получить результат в виде формулы (21), а не (22).

При составлении такого алгоритма прежде всего нас будут интересовать подробности реализации этого процесса внутри компьютера. Во многих языках высокого уровня и в специальных программах, которые доступны для большинства типов компьютеров, предусмотрены специальные встроенные средства, позволяющие упростить подобные алгебраические преобразования. Однако назначение этого примера заключается в приобретении опыта работы с основными операциями с деревьями.

Основная идея такого алгоритма заключается в обходе дерева в обратном порядке с вычислением производной каждого посещаемого узла до тех пор, пока не будет вычислена производная всего выражения. Использование обратного порядка обхода означает, что узел оператора (например, "+") посещается *после* того, как будут продифференцированы его операнды. Правила (11)-(19) подразумевают, что

каждая подчиненная формула исходной формулы рано или поздно будет продифференцирована, а потому дифференцирование можно выполнять в обратном порядке обхода дерева.

При работе с правопрошитым деревом уже необязательно применять стек во время выполнения этого алгоритма. С другой стороны, недостаток подобного представления дерева заключается в том, что необходимо делать копии поддеревьев. Например, в правиле для $D(u \uparrow v)$ потребуется трижды копировать u и v, тогда как вместо дерева можно было бы использовать представление в виде Списка из раздела 2.3.5 и избежать такого копирования.

Алгоритм D (*Дифференцирование*). Если `Y` — адрес заголовка списка, который указывает на формулу, представленную в описанном выше виде, а `DY` — адрес заголовка списка для пустого дерева, то в результате выполнения этого алгоритма `NODE(DY)` будет указывать на дерево, представляющее производную от `Y` по `X`.

D1. [Инициализация.] Установить `P` $\leftarrow$ `Y$` (а именно, первый узел дерева при обходе в обратном порядке, который является первым узлом соответствующего бинарного дерева в симметричном порядке).

D2. [Дифференцирование.] Установить `P1` $\leftarrow$ `LLINK(P)` и, если `P1` $\neq \Lambda$, также установить `Q1` $\leftarrow$ `RLINK(P1)`. Затем выполнить описанную ниже программу `DIFF[TYPE(P)]`. (Программы `DIFF[0]` `DIFF[1]` и т. д. вычисляют производную дерева с корнем `P` и задают для указательной переменной `Q` значение адреса корня этой производной. Чтобы упростить спецификацию программ `DIFF`, прежде всего следует установить переменные `P1` и `Q1`.)

D3. [Восстановление связи.] Если `TYPE(P)` обозначает бинарный оператор, установить `RLINK(P1)` $\leftarrow$ `P2`. (Пояснение дается ниже, при описании следующего шага.)

D4. [Продвижение к `P$`.] Установить `P2` $\leftarrow$ `P`, `P` $\leftarrow$ `P$`. Теперь, если `RTAG(P2)` $= 0$ (т. е. если `NODE(P2)` имеет справа брата или сестру), установить `RLINK(P2)` $\leftarrow$ `Q`. (Именно здесь и заключена суть этого алгоритма: структура дерева `Y` временно разрушается так, чтобы сохранить для дальнейшего использования связь с производной `P2`. Недостающая связь будет восстановлена позднее, на шаге D3. Более подробное описание этой уловки приводится в упр. 21.)

D5. [Обход завершен?] Если `P` $\neq$ `Y`, вернуться к шагу D2. В противном случае установить `LLINK(DY)` $\leftarrow$ `Q` и `RLINK(Q)` $\leftarrow$ `DY`, `RTAG(Q)` $\leftarrow 1$. ∎

Программа, описанная в алгоритме D, является общей программой операций дифференцирования, непосредственно выполняемых программами `DIFF[0]`, `DIFF[1]`, ..., которые вызываются на шаге D2. Во многом алгоритм D напоминает программу управления интерпретирующей системы (или компилятор), которая рассмотрена в разделе 1.4.3, но совершает обход дерева, а не простой последовательности инструкций.

Для завершения алгоритма D необходимо определить программы, которые непосредственно выполняют дифференцирование. Далее утверждение "`P` указывает на дерево" будет означать, что узел `NODE(P)` является корнем дерева, которое хранится в памяти компьютера в виде правопрошитого бинарного дерева, хотя оба указателя `RLINK(P)` и `RTAG(P)` утрачивают смысл при работе с таким деревом. Воспользуемся *функцией конструирования деревьев* (*tree construction function*), которая

позволяет создавать новые деревья за счет объединения деревьев меньшего размера. Пусть x обозначает узел некоторого типа, содержащий константу, переменную или оператор, а U и V обозначают указатели на деревья. В таком случае

TREE(x,U,V) образует новое дерево с корнем x и поддеревьями U и V этого корня: W ⇐ AVAIL, INFO(W) ← x, LLINK(W) ← U, RLINK(U) ← V, RTAG(U) ← 0, RLINK(V) ← W, RTAG(V) ← 1;

TREE(x,U) аналогично образует новое дерево только с одним поддеревом: W ⇐ AVAIL, INFO(W) ← x, LLINK(W) ← U, RLINK(U) ← W, RTAG(U) ← 1;

TREE(x) образует новое дерево с корнем x, который является концевым узлом: W ⇐ AVAIL, INFO(W) ← x, LLINK(W) ← Λ.

Кроме того, для TYPE(W) задается значение, соответствующее типу узла x. Во всех случаях значением TREE является W, т. е. указатель на только что построенное дерево. Читателю следует тщательно изучить эти определения, поскольку они иллюстрируют представление дерева на основе бинарного дерева. Еще одна функция, COPY(U), создает копию дерева, на которое указывает U, а ее значением является указатель на только что созданное дерево. Основные функции TREE и COPY упрощают процесс поэтапного создания дерева, соответствующего производной для заданной формулы.

Нуль-арные операторы (*константы и переменные*). Для этих операций узел NODE(P) является концевым, а значения переменных P1, P2, Q1 и Q для их выполнения несущественны.

DIFF[0]: (NODE(P) — константа.) Установить Q ← TREE(0).

DIFF[1]: (NODE(P) — переменная.) Если INFO(P) = "X", установить Q ← TREE(1); в противном случае установить Q ← TREE(0).

Унарные операторы (*логарифм и отрицание*). Для этих операций NODE(P) имеет одного ребенка, U, на которого указывает P1, а Q указывает на $D(U)$. Значения переменных P2 и Q1 для выполнения этих операций несущественны.

DIFF[2]: (NODE(P) — "ln".) Если INFO(Q) ≠ 0, установить Q ← TREE("/",Q, COPY(P1)).

DIFF[3]: (NODE(P) — "neg".) Если INFO(Q) ≠ 0, установить Q ← TREE("neg",Q).

Бинарные операторы (*сложение, вычитание, умножение, деление, возведение в степень*). Для этих операций NODE(P) имеет двух детей, U и V, на которых указывают P1 и P2 соответственно; Q1 и Q указывают на $D(U)$ и $D(V)$ соответственно.

DIFF[4]: (Операция "+".) Если INFO(Q1) = 0, установить AVAIL ⇐ Q1. В противном случае, если INFO(Q) = 0, установить AVAIL ⇐ Q и Q ← Q1; иначе — установить Q ← TREE("+",Q1,Q).

DIFF[5]: (Операция "−".) Если INFO(Q) = 0, установить AVAIL ⇐ Q и Q ← Q1. В противном случае, если INFO(Q1) = 0, установить AVAIL ⇐ Q1 и Q ← TREE("neg",Q); иначе — установить Q ← TREE("−",Q1,Q).

DIFF[6]: (Операция "×".) Если INFO(Q1) ≠ 0, установить Q1 ← MULT(Q1, COPY(P2)). Затем, если INFO(Q) ≠ 0, установить Q ← MULT(COPY(P1),Q). После этого перейти к выполнению программы DIFF[4].

Здесь MULT(U,V) является новой функцией, которая создает дерево для U × V и проверяет, не равны ли U или V единице:

если INFO(U) = 1 и TYPE(U) = 0, установить AVAIL ⇐ U и MULT(U,V) ← V;
если INFO(V) = 1 и TYPE(V) = 0, установить AVAIL ⇐ V и MULT(U,V) ← U;
в противном случае установить MULT(U,V) ← TREE("×",U,V).

DIFF[7]: (Операция "/".) Если INFO(Q1) ≠ 0, установить

$$\texttt{Q1} \leftarrow \texttt{TREE("/",Q1,COPY(P2))}.$$

Затем, если INFO(Q) ≠ 0, установить

$$\texttt{Q} \leftarrow \texttt{TREE("/",MULT(COPY(P1),Q),TREE("↑", COPY(P2),TREE(2)))}.$$

После этого перейти к выполнению программы DIFF[5].

DIFF[8]: (Операция "↑".) См. упр. 12.

В заключение настоящего раздела продемонстрируем применение этих операций в компьютерной программе на основе только внутреннего языка компьютера MIX.

Программа D (*Дифференцирование*). Приведенная ниже программа на языке MIXAL реализует алгоритм D с такими значениями регистров rI1 ≡ P, rI3 ≡ P2, rI4 ≡ P1, rI5 ≡ Q, rI6 ≡ Q1. Для удобства порядок вычислений немного изменен.

```
001  * ДИФФЕРЕНЦИРОВАНИЕ В ПРАВОПРОШИТОМ ДЕРЕВЕ
002  LLINK  EQU  4:5          Определение полей, см. (10).
003  RLINK  EQU  1:2
004  RLINKT EQU  0:2
005  TYPE   EQU  3:3
006  * УПРАВЛЯЮЩАЯ ПРОГРАММА  D1. Инициализация.
007  D1     STJ  9F           Эта программа рассматривается как подпрограмма.
008         LD4  Y(LLINK)     P1 ← LLINK(Y), приготовиться к поиску Y$.
009  1H     ENT2 0,4          P ← P1.
010  2H     LD4  0,2(LLINK)   P1 ← LLINK(P).
011         J4NZ 1B           Если P1 ≠ Λ, повторить.
012  D2     LD1  0,2(TYPE)    D2. Дифференцирование.
013         JMP  *+1,1        Переход к DIFF[TYPE(P)].
014         JMP  CONSTANT     Переход к элементу таблицы DIFF[0].
015         JMP  VARIABLE                                DIFF[1].
016         JMP  LN                                      DIFF[2].
017         JMP  NEG                                     DIFF[3].
018         JMP  ADD                                     DIFF[4].
019         JMP  SUB                                     DIFF[5].
020         JMP  MUL                                     DIFF[6].
021         JMP  DIV                                     DIFF[7].
022         JMP  PWR                                     DIFF[8].
023  D3     ST3  0,4(RLINK)   D3. Восстановление связи. RLINK(P1) ← P2.
024  D4     ENT3 0,2          D4. Продвижение к P$. P2 ← P.
025         LD2  0,2(RLINKT)  P ← RLINKT(P).
026         J2N  1F           Переход, если RTAG(P) = 1;
027         ST5  0,3(RLINK)       в противном случае установить RLINK(P2) ← Q.
028         JMP  2B           Обратите внимание, что узел NODE(P$) — концевой.
029  1H     ENN2 0,2
```

```
030  D5      ENT1 -Y,2          D5. Обход завершен?
031          LD4  0,2(LLINK)    P1 ← LLINK(P), приготовиться к шагу D2.
032          LD6  0,4(RLINK)    Q1 ← RLINK(P1)
033          J1NZ D2            Переход к шагу D2, если P ≠ Y;
034          ST5  DY(LLINK)         в противном случае установить LLINK(DY) ← Q.
035          ENNA DY
036          STA  0,5(RLINKT)   RLINK(Q) ← DY, RTAG(Q) ← 1.
037  9H      JMP  *             Выход из программы дифференцирования. ∎
```

В следующей части программы содержатся основные подпрограммы TREE и COPY. Первая имеет три входа: TREE0, TREE1 и TREE2, в соответствии с количеством поддеревьев создаваемого дерева. Независимо от того, какой вход в подпрограмму используется, регистр rA будет содержать особую константу, которая указывает тип узла-корня конструируемого дерева. Эти особые константы представлены в строках 105–124.

```
038  * ОСНОВНЫЕ ПОДПРОГРАММЫ КОНСТРУИРОВАНИЯ ДЕРЕВА
039  TREE0               STJ  9F            Функция TREE(rA).
040                      JMP  2F
041  TREE1               ST1  3F(0:2)       Функция TREE(rA,rI1).
042                      JSJ  1F
043  TREE2               STX  3F(0:2)       Функция TREE(rA,rX,rI1).
044  3H                  ST1  *(RLINKT)     RLINK(rX) ← rI1, RTAG(rX) ← 0.
045  1H                  STJ  9F
046                      LDXN AVAIL
047                      JXZ  OVERFLOW
048                      STX  0,1(RLINKT)   RLINK(rI1) ← AVAIL, RTAG(rI1) ← 1.
049                      LDX  3B(0:2)
050                      STA  *+1(0:2)
051                      STX  *(LLINK)      Установить LLINK для узла,
                                                следующего за корнем.
052  2H                  LD1  AVAIL         rI1 ⇐ AVAIL.
053                      J1Z  OVERFLOW
054                      LDX  0,1(LLINK)
055                      STX  AVAIL
056                      STA  *+1(0:2)      Копировать корень в новый узел.
057                      MOVE *(2)
058                      DEC1 2             Переустановить rI1, чтобы он
                                                указывал на новый корень.
059  9H                  JMP  *             Выход из функции TREE,
                                                rI1 указывает на новое дерево.
060  COPYP1              ENT1 0,4           COPY(P1), особый вход в COPY.
061                      JSJ  COPY
062  COPYP2              ENT1 0,3           COPY(P2), особый вход в COPY.
063  COPY                STJ  9F            Функция COPY(rI1).
 ⋮                       ⋮                 (См. упр. 13 )
104  9H                  JMP  *             Выход из COPY,
                                                rI1 указывает на новое дерево.
105  CON0                CON  0             Узел, представляющий "0".
106                      CON  0
107  CON1                CON  0             Узел, представляющий "1".
```

```
108                 CON  1
109  CON2           CON  0          Узел, представляющий "2".
110                 CON  2
111  LOG            CON  2(TYPE)    Узел, представляющий "ln".
112                 ALF    LN
113  NEGOP          CON  3(TYPE)    Узел, представляющий "neg".
114                 ALF   NEG
115  PLUS           CON  4(TYPE)    Узел, представляющий "+".
116                 ALF    +
117  MINUS          CON  5(TYPE)    Узел, представляющий "−".
118                 ALF
119  TIMES          CON  6(TYPE)    Узел, представляющий "×".
120                 ALF    *
121  SLASH          CON  7(TYPE)    Узел, представляющий "/".
122                 ALF    /
123  UPARROW        CON  8(TYPE)    Узел, представляющий "↑".
124                 ALF    **       ▮
```

Оставшаяся часть программы соответствует программам дифференцирования DIFF[0], DIFF[1], ...; эти программы задуманы так, что после обработки бинарного оператора они возвращают управление шагу D3, а в противном случае — шагу D4.

```
125  * ПРОГРАММЫ ДИФФЕРЕНЦИРОВАНИЯ
126  VARIABLE   LDX  1,2
127             ENTA CON1
128             CMPX 2F          Верно ли, что INFO(P) = "X"?
129             JE   *+2         Если верно, вызвать функцию TREE(1).
130  CONSTANT   ENTA CON0        Вызвать функцию TREE(0).
131             JMP  TREE0
132  1H         ENT5 0,1         Q ← адрес нового дерева.
133             JMP  D4          Возврат к управляющей программе.
134  2H         ALF      X
135  LN         LDA  1,5
136             JAZ  D4          Возврат к управляющей программе,
137             JMP  COPYP1          если INFO(Q) = 0; в противном случае
138             ENTX 0,5             установить rI1 ← COPY(P1).
139             ENTA SLASH
140             JMP  TREE2       rI1 ← TREE("/",Q,rI1).
141             JMP  1B          Q ← rI1, возврат к управляющей программе.
142  NEG        LDA  1,5
143             JAZ  D4          Если INFO(Q) = 0, возврат.
144             ENTA NEGOP
145             ENT1 0,5
146             JMP  TREE1       rI1 ← TREE("neg",Q).
147             JMP  1B          Q ← rI1, возврат к управляющей программе.
148  ADD        LDA  1,6
149             JANZ 1F          Переход, если не выполняется INFO(Q1) = 0.
150  3H         LDA  AVAIL       AVAIL ⇐ Q1.
151             STA  0,6(LLINK)
152             ST6  AVAIL
153             JMP  D3          Возврат к управляющей программе,
154  1H         LDA  1,5             бинарный оператор.
```

```
155             JANZ 1F          Переход, если не выполняется INFO(Q) = 0
156  2H         LDA  AVAIL       AVAIL ⇐ Q
157             STA  0,5(LLINK)
158             ST5  AVAIL
159             ENT5 0,6         Q ← Q1
160             JMP  D3          Возврат к управляющей программе
161  1H         ENTA PLUS        Подготовиться к вызову TREE("+",Q1,Q)
162  4H         ENTX 0,6
163             ENT1 0,5
164             JMP  TREE2
165             ENT5 0,1         Q ← TREE("±",Q1,Q)
166             JMP  D3          Возврат к управляющей программе
167  SUB        LDA  1,5
168             JAZ  2B          Переход, если INFO(Q) = 0.
169             LDA  1,6
170             JANZ 1F          Переход, если не выполняется INFO(Q1) = 0
171             ENTA NEGOP
172             ENT1 0,5
173             JMP  TREE1
174             ENT5 0,1         Q ← TREE("neg",Q)
175             JMP  3B          AVAIL ⇐ Q1 и возврат.
176  1H         ENTA MINUS       Подготовиться к вызову TREE("−",Q1,Q)
177             JMP  4B
178  MUL        LDA  1,6
179             JAZ  1F          Переход, если INFO(Q1) = 0,
180             JMP  COPY2           в противном случае rI1 ← COPY(P2)
181             ENTA 0,6
182             JMP  MULT        rI1 ← MULT(Q1,COPY(P2)).
183             ENT6 0,1         Q1 ← rI1
184  1H         LDA  1,5
185             JAZ  ADD         Переход, если INFO(Q) = 0,
186             JMP  COPYP1          в противном случае rI1 ← COPY(P1).
187             ENTA 0,1
188             ENT1 0,5
189             JMP  MULT        rI1 ← MULT(COPY(P1),Q)
190             ENT5 0,1         Q ← rI1
191             JMP  ADD
192  MULT       STJ  9F          Подпрограмма MULT(rA,rI1).
193             STA  1F(0:2)     Пусть rA ≡ U, rI1 ≡ V
194             ST2  8F(0:2)     Сохранить rI2
195  1H         ENT2 *           rI2 ← U
196             LDA  1,2         Проверить, верно ли, что INFO(U) = 1,
197             DECA 1
198             JANZ 1F
199             LDA  0,2(TYPE)       и верно ли, что TYPE(U) = 0
200             JAZ  2F
201  1H         LDA  1,1         Если не верно, проверить, верно ли INFO(V) = 1
202             DECA 1
203             JANZ 1F
204             LDA  0,1(TYPE)       и верно ли, что TYPE(V) = 0.
205             JANZ 1F
```

```
206             ST1  *+2(0:2)      Если верно, выполнить обмен U ↔ V.
207             ENT1 0,2
208             ENT2 *
209  2H         LDA  AVAIL         AVAIL ⇐ U.
210             STA  0,2(LLINK)
211             ST2  AVAIL
212             JMP  8F            В результате получим V.
213  1H         ENTA TIMES
214             ENTX 0,2
215             JMP  TREE2         В результате получим TREE("×",U,V).
216  8H         ENT2 *             Восстановить rI2.
217  9H         JMP  *             Выход из MULT с результатом в rI1.  ▮
```

Две другие программы DIV и PWR выглядят аналогично, и читателю предлагается самостоятельно создать их в качестве упражнения (см. упр. 15 и 16).

УПРАЖНЕНИЯ

▶ **1.** [*20*] В этом разделе приводилось формальное определение бинарного дерева $B(F)$, соответствующего лесу F. Дайте формальное определение с обратным смыслом, т. е. определите лес $F(B)$, который соответствует бинарному дереву B.

▶ **2.** [*20*] Обозначение дерева в десятичной системе обозначений Дьюи дано в разделе 2.3, а обозначение бинарных деревьев — в упр. 2.3.1–5. Таким образом, узел "J" в (1) представлен в виде "2.2.1", а в эквивалентном бинарном дереве (3) — в виде "11010". Если это возможно, предложите правило, которое непосредственно выражает естественное соответствие между деревьями и бинарными деревьями на основе десятичной системы обозначений Дьюи.

3. [*22*] Какая существует связь между десятичной системой обозначений Дьюи для узлов леса и прямым и обратным порядками обхода этих узлов?

4. [*19*] Справедливо ли следующее утверждение: "Концевые узлы дерева встречаются в одних и тех же относительных позициях при обходе как в прямом, так и в обратном порядках"?

5. [*23*] Другое соответствие леса и бинарных деревьев можно определить с помощью RLINK(P), который указывает на крайнего справа ребенка узла NODE(P), а также с помощью LLINK(P), который указывает на крайнего слева ребенка узла. Пусть F является лесом, который, таким образом, соответствует бинарному дереву B. Какой порядок узлов бинарного дерева B соответствует (a) прямому и (b) обратному порядкам обхода леса F?

6. [*25*] Пусть T *является непустым бинарным деревом, в котором каждый узел имеет* 0 или 2 детей. Если рассматривать T как бинарное дерево, оно соответствует (на основе естественного соответствия) *другому* бинарному дереву T'. Существует ли какая-либо простая связь между прямым, симметричным и обратным порядками обхода узлов дерева T (согласно их определениям для бинарных деревьев) и теми же порядками обхода узлов дерева T'?

7. [*M20*] Лес может рассматриваться как частично упорядоченный, если считать, что каждый узел дерева расположен перед его последователями. Можно ли утверждать, что узлы рассортированы в топологическом порядке (согласно определению в разделе 2.2.3), если они находятся: (a) в прямом порядке, (b) в обратном порядке, (c) в реверсивном прямому порядке и (d) в реверсивном обратному порядке?

8. [*M20*] В упр. 2.3.1–25 показано, как порядок следования данных в отдельных узлах бинарного дерева может быть расширен до линейного упорядочения всех бинарных деревьев. При наличии естественного соответствия аналогично можно получить упорядочение всех деревьев. Переформулируйте данное в этом упражнении определение для деревьев.

9. [*M21*] Покажите, что существует простая связь между общим количеством неконцевых узлов леса и общим количеством правых связей, которые равны Λ в соответствующем непрошитом бинарном дереве.

10. [*M23*] Пусть F — это лес деревьев, узлы которых упорядочены в прямом порядке обхода $u_1, u_2, \ldots, u_n$, а F' — лес деревьев, узлы которых упорядочены в прямом порядке обхода $u'_1, u'_2, \ldots, u'_{n'}$. Введем обозначение $d(u)$ для степени (количества детей) узла u. На основании этих понятий сформулируйте и докажите теорему, аналогичную теореме 2.3.1А.

11. [*15*] Нарисуйте схему деревьев, аналогичную схеме (7), которая соответствует формуле $y = e^{-x^2}$.

12. [*M21*] Предложите спецификации для программы `DIFF[8]` (операция "↑"), которые опущены при описании алгоритма дифференцирования в данном разделе.

▸ **13.** [*26*] Создайте для компьютера `MIX` подпрограмму `COPY` (которая в программе из данного раздела находится в строках 063–104). [*Подсказка.* Адаптируйте алгоритм 2.3.1С для правопрошитого бинарного дерева с соответствующими начальными условиями.]

▸ **14.** [*M21*] Сколько времени потребуется программе из упр. 13 для копирования дерева с n узлами?

15. [*23*] Создайте для компьютера `MIX` подпрограмму `DIV`, соответствующую `DIFF[7]`. (Эта программа располагается в строке 217 описанной в разделе программы.)

16. [*24*] Создайте для компьютера `MIX` подпрограмму `PWR`, соответствующую `DIFF[8]` из упр. 12. (Эта программа располагается после программы, приведенной в решении к упр. 15.)

17. [*M40*] Создайте программу для упрощения алгебраических выражений, которая позволяет упростить, например, выражения (20) и (21) и привести их к виду (22). [*Указание.* Включите в каждый узел новое поле, которое представляет его коэффициент (для слагаемых) или степень (для сомножителей). Примените такие алгебраические тождества, как замена выражения $\ln(u \uparrow v)$ выражением $v \ln u$ сокращения операторов $-$, $/$, $\uparrow$ и neg за счет выполнения эквивалентных им операций сложения и умножения тогда, когда это только возможно. Преобразуйте бинарные операторы "+" и "×" в n-арные. Приведите подобные члены, сортируя их операнды в древовидном порядке (упр. 8). Некоторые суммы и произведения сократятся до нуля или единицы, позволяя выполнить дальнейшие упрощения. Разумеется, что здесь следует использовать и другие упрощения, например сумму логарифмов можно заменить логарифмом произведения.]

▸ **18.** [*25*] Ориентированное дерево, указанное с помощью n связей `PARENT`$[j]$ для $1 \le j \le n$, неявно определяет упорядоченное дерево, если узлы в каждой семье упорядочены по адресам. Создайте эффективный алгоритм, который конструирует дважды связанный циклический список, содержащий узлы этого упорядоченного дерева в прямом порядке обхода. Например, при условии

$$\begin{aligned} j &= 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8 \\ \texttt{PARENT}[j] &= 3\ 8\ 4\ 0\ 4\ 8\ 3\ 4 \end{aligned}$$

такой алгоритм позволяет получить

$$\begin{aligned} \texttt{LLINK}[j] &= 3\ 8\ 4\ 6\ 7\ 2\ 1\ 5 \\ \texttt{RLINK}[j] &= 7\ 6\ 1\ 3\ 8\ 4\ 5\ 2 \end{aligned}$$

а также указать, что корневым является узел 4.

19. [*M35*] Назовем *свободной решеткой* (*free lattice*) математическую систему, которая (в этом упражнении) может быть достаточно просто определена как множество формул, состоящих из переменных и двух абстрактных бинарных операторов "$\vee$" и "$\wedge$". Отношение "$X \succeq Y$" определяется в свободной решетке между некоторыми формулами X и Y согласно следующим правилам:

i) $X \vee Y \succeq W \wedge Z$ тогда и только тогда, когда $X \vee Y \succeq W$ или $X \vee Y \succeq Z$, или $X \succeq W \wedge Z$, или $Y \succeq W \wedge Z$,

ii) $X \wedge Y \succeq Z$ тогда и только тогда, когда $X \succeq Z$ и $Y \succeq Z$;

iii) $X \succeq Y \vee Z$ тогда и только тогда, когда $X \succeq Y$ и $X \succeq Z$;

iv) $x \succeq Y \wedge Z$ тогда и только тогда, когда $x \succeq Y$ или $x \succeq Z$, где x является переменной;

v) $X \vee Y \succeq z$ тогда и только тогда, когда $X \succeq z$ или $Y \succeq z$, где z является переменной,

vi) $x \succeq y$ тогда и только тогда, когда $x = y$, где x и y являются переменными.

Например, находим $a \wedge (b \vee c) \succeq (a \wedge b) \vee (a \wedge c) \not\succeq a \wedge (b \vee c)$

Создайте алгоритм, с помощью которого можно установить, выполняется ли отношение $X \succeq Y$ для двух заданных формул X и Y в свободной решетке

▸ **20.** [*M22*] Докажите, что если u и v — узлы леса, то узел u является предком узла v тогда и только тогда, когда u располагается перед узлом v в прямом порядке обхода и u располагается за узлом v в обратном порядке.

21. [*25*] Алгоритм D управляет процессом дифференцирования выражений с бинарными, унарными и нуль-арными операторами, которые могут быть представлены деревьями с узлами со степенями 2, 1 и 0. Но в нем явным образом не указан способ управления тернарными операторами и операторами с более высокой степенью. (Например, в упр. 17 предполагается, что операции сложения и умножения выполняются для любого количества операндов.) Можно ли предложить достаточно простой способ расширения алгоритма D, чтобы его можно было применять для дифференцирования выражений с операторами, узлы которых имеют степени выше 2?

▸ **22.** [*M26*] Положим, дерево T *может быть вставлено* в дерево T', что обозначается как $T \subseteq T'$, если существует такое взаимно однозначное отображение f узлов дерева T на узлы дерева T', при котором сохраняются прямой и обратный порядки. (Иначе говоря, u предшествует v при прямом порядке обхода дерева T тогда и только тогда, когда узел $f(u)$ предшествует узлу $f(v)$ в прямом порядке обхода узлов дерева T', причем то же самое верно и для обратного порядка обхода (рис. 25).)

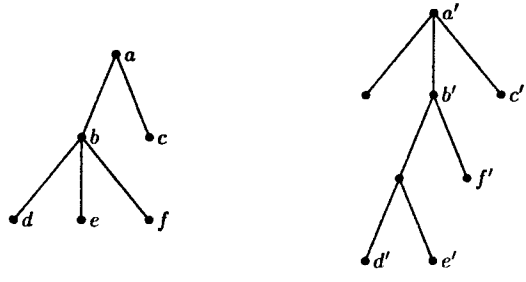


Рис. 25. Пример одного дерева, вставленного в другое.

Если T имеет более одного узла, допустим, что $l(T)$ является крайним слева поддеревом корня дерева T, а $r(T)$ — остальной частью дерева T, т. е. деревом T без $l(T)$.

Докажите, что T может быть вставлено в дерево T', если (i) T имеет только один узел или (ii) оба дерева T и T' имеют более одного узла и либо $T \subseteq l(T')$, либо $T \subseteq r(T')$, или $(l(T) \subseteq l(T')$ и $r(T) \subseteq r(T'))$. Справедливо ли обратное утверждение?

2.3.3. Другие представления деревьев

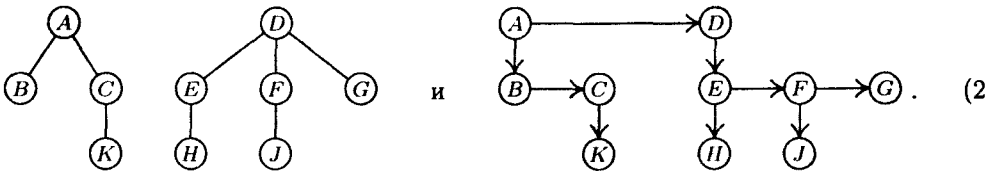
Кроме описанного в предыдущем разделе метода, основанного на указателях LLINK–RLINK (левый ребенок — правый брат (или сестра)), существует много других способов представления древовидных структур. Как обычно, наиболее подходящий способ в значительной мере зависит от типа операций, выполняемых над этими деревьями. В данном разделе рассматривается несколько методов представления деревьев, которые на практике доказали свою целесообразность.

Сначала рассмотрим методы *последовательного* распределения памяти. Как и в случае линейных списков, этот способ распределения наиболее удобен для компактного представления древовидной структуры, размер и форма которой во время выполнения программы не претерпевает значительных динамических изменений. Существует множество ситуаций, в которых для организации ссылок внутри программы необходимо использовать именно постоянные таблицы данных с древовидной структурой. А необходимая форма этих деревьев в памяти компьютера зависит от способа доступа к данным из этой таблицы.

Наиболее популярный тип последовательного представления деревьев (и лесов) соответствует устранению полей LLINK и использованию вместо них метода последовательной адресации. Рассмотрим, например, следующий лес, который уже упоминался в предыдущем разделе:

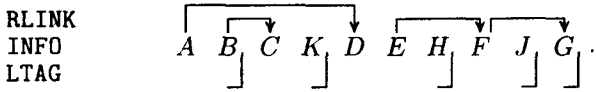
$$\big(A(B,C(K)),\ D(E(H),F(J),G)\big). \tag{1}$$

Схематически его можно представить так:



(2)

При использовании *последовательного представления в прямом порядке* (*preorder sequential representation*) узлы располагаются в прямом порядке с полями INFO, RLINK и LTAG в каждом узле:



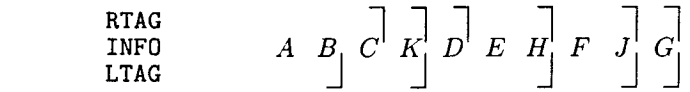
(3)

Здесь непустые связи RLINK обозначены стрелками, а LTAG = 1 (для концевых узлов) — символом "⌋". Связи LLINK не нужны, поскольку они либо пусты, либо указывают на следующий объект последовательности. Читателю будет полезно сравнить (1) с (3).

Это представление обладает сразу несколькими интересными свойствами. Во-первых, все поддеревья узла располагаются сразу за самим узлом, а потому все

поддеревья внутри исходного леса находятся в последовательных блоках. [Сравните это с представлением на основе "вложенных скобок" в (1) и на рис. 20, (b).] Во-вторых, обратите внимание на то, что стрелки связей RLINK на схеме (3) никогда не пересекаются. Это справедливо и в общем случае, так как в бинарном дереве все узлы, находящиеся между X и RLINK(X), при прямом порядке обхода располагаются в левом поддереве X, а потому из этой части дерева не выходят никакие стрелки. В-третьих, можно заметить, что поле LTAG, которое указывает, будет ли узел концевым, является лишним, так как символ "⌋" располагается только в конце леса и только *перед* каждой направленной вниз стрелкой.

Действительно, из этих замечаний следует, что даже поле RLINK также почти излишне и на самом деле для представления такой структуры необходимы поля RTAG и LTAG. Следовательно, схему (3) можно получить на основе меньшего объема данных:

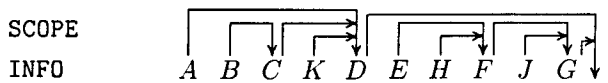


(4)

При считывании представления (4) слева направо узлы с полями RTAG $\neq$ "⌉" соответствуют непустым значениям RLINK. Тогда для восстановления связей RLINK нужно каждый раз после прохождения узла с полем LTAG = "⌋" проводить связь к текущему узлу от ближайшего предшествующего узла с невосстановленной связью RLINK. (Например, после прохождения узла B с LTAG = "⌋" связь к текущему узлу C следует проводить от узла B. затем после прохождения узла K связь к текущему узлу D необходимо проводить от узла A (поскольку связь RLINK в узле B уже восстановлена) и т. д. — *Прим. перев.*) Следовательно, ячейки с невосстановленными, т. е. ненулевыми, значениями связей RLINK можно хранить в стеке. Таким образом, снова доказана теорема 2.3.1A.

Тот факт, что поля RLINK и LTAG являются избыточными в представлении (3), не имеет большого значения, за исключением случая, когда нужно полностью выполнить последовательный обход всего дерева, поскольку для получения недостающей информации потребуются дополнительные вычисления. Значит, чаще всего нужны все данные представления (3). Однако очевидно, что при этом значительная часть памяти расходуется очень неэкономно, например в рассмотренном здесь частном примере леса больше половины полей RLINK равны Λ. Для более эффективного использования памяти можно прибегнуть к следующим двум распространенным способам.

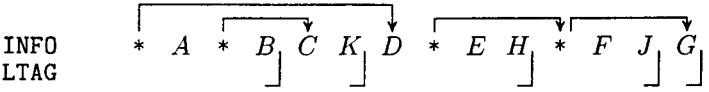
1) В поле RLINK каждого узла указать адрес, следующий за всеми узлами поддерева данного узла. Это поле часто называется "SCOPE" (т. е. область действия), а не RLINK, поскольку оно обозначает правую границу "влияния" (на наследников) каждого узла. Теперь вместо схемы (3) получим другую схему, в которой стрелки также не пересекаются:



(5)

Более того, LTAG(X) = "⌋" характеризуется условием SCOPE(X) = X + c, где c — количество слов в узле. Пример использования понятия SCOPE приводится в упр. 2.4 12.

2) Уменьшить размер каждого узла, удалив поле RLINK, и добавить особые "узлы связи" непосредственно перед узлами, которые прежде содержали непустые связи RLINK:



(6)

Здесь символ "*" обозначает такие особые узлы связи, в которых поля INFO каким-то образом характеризуют их, как связи, направления которых указаны стрелками. Если поля INFO и RLINK в схеме (3) занимают приблизительно одинаковый объем памяти, то переход к схеме (6), в общем, позволяет сэкономить место в памяти, так как количество узлов "*" всегда меньше количества других узлов (т. е. тех узлов, которые не являются особыми узлами связи "*"). В некоторой степени представление (6) аналогично последовательности команд в одноадресном компьютере, подобном компьютеру MIX с узлами "*", которые соответствуют командам условного перехода.

Другой вид последовательного распределения, аналогичного (3), может быть получен за счет удаления полей RLINK, а не полей LLINK. В этом случае узлы леса могут быть перечислены в новом порядке, который называется *фамильным порядком* (*family order*), поскольку члены каждой "семьи" располагаются рядом. Фамильный порядок для любого леса может быть получен рекурсивно так, как показано ниже.

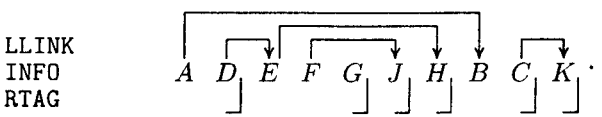
Посетить корень первого дерева.

Совершить обход остальных деревьев (в фамильном порядке).

Совершить обход поддеревьев корня первого дерева (в фамильном порядке).

(Сравните этот способ с определениями прямого и обратного порядков из предыдущего раздела. Фамильный порядок идентичен инверсивному обратному порядку обхода соответствующего бинарного дерева)

Последовательное представление фамильного порядка (*family order sequential representation*) деревьев (2) выглядит так:

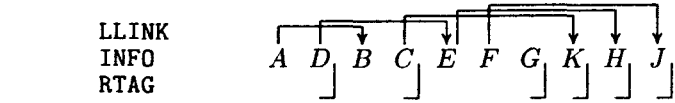


(7)

В этом случае поля RTAG служат для разделения семей. При фамильном порядке сначала перечисляются корни всех деревьев в лесу, а затем — отдельные семьи. Причем каждый раз выбор семей начинается с ближайшего перечисленного узла, семья которого еще не перечислялась. Из этого следует, что стрелки LLINK никогда не пересекаются, а другие свойства представления прямого порядка переносятся аналогичным образом.

Можно было бы не использовать фамильный порядок, а просто перечислить узлы слева направо, последовательно уровень за уровнем. Такой способ называется порядком уровней (level order) [см G. Salton, *CACM* **5** (1962). 103–114]. *Последовательное представление порядка уровней* (*level order sequential representation*) для (2)

выглядит так:



(8)

Оно подобно представлению (7), но семьи выбраны в порядке "первым вошел — первым вышел", а не в порядке "последним вошел — первым вышел". Представления деревьев в виде (7) или (8) могут рассматриваться как естественный аналог для деревьев последовательного представления линейных списков.

Читатель легко сообразит, как можно создать алгоритм обхода и анализа деревьев, которые показаны выше, в последовательном представлении, поскольку информация из полей LLINK и RLINK позволяет рассматривать эти структуры как полностью связанную древовидную структуру.

Еще один метод, который называется *обратным порядком со степенями* (*postorder with degrees*), несколько отличается от описанных выше методов. В случае его использования узлы перечисляются в обратном порядке и вместо связей для каждого узла указывается его степень:

$$\begin{array}{rcccccccccc} \texttt{DEGREE} & 0 & 0 & 1 & 2 & 0 & 1 & 0 & 1 & 0 & 3 \\ \texttt{INFO} & B & K & C & A & H & E & J & F & G & D \end{array} \qquad (9)$$

Доказательство достаточности этих сведений для характеристики древовидной структуры приводится в упр 2.3.2–10. Такой порядок очень удобен для оценки "снизу вверх" (bottom-up) значений функций, заданных в узлах дерева так, как показано в следующем алгоритме.

Алгоритм F (*Оценка функции, локально определенной в узлах дерева*). Пусть f — такая функция узлов дерева, что значение f в узле x зависит только от x и значений f для детей узла x. Данный алгоритм позволяет с помощью вспомогательного стека оценить f в каждом узле непустого леса.

F1. [Инициализация.] Установить стек пустым, а P пусть указывает на первый узел леса в обратном порядке.

F2. [Оценка f.] Установить $d \leftarrow$ DEGREE(P). (При первой попытке выполнения этого шага значение d будет равно нулю. Вообще, по достижении данного шага верхними элементами d стека по направлению сверху вниз будут элементы $f(x_d)$, ..., $f(x_1)$, где x_1, ..., x_d — дети узла NODE(P) слева направо.) Оценить $f\big($NODE(P)$\big)$, используя значения стека $f(x_d)$, ..., $f(x_1)$.

F3. [Обновление стека.] Удалить из стека верхние d элементов, а затем разместить значение $f\big($NODE(P)$\big)$ в верхней части стека.

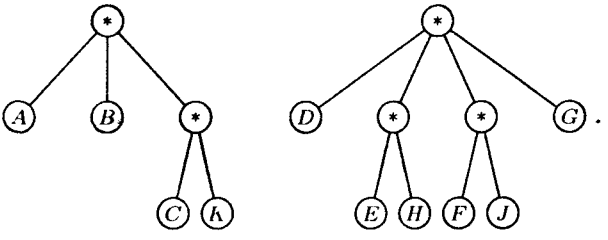
F4. [Продвижение.] Если P — последний узел в обратном порядке обхода, то прекратить выполнение алгоритма. (Тогда стек будет содержать следующие элементы по направлению сверху вниз: $f\big($корень$(T_m)\big)$, ..., $f\big($корень$(T_1)\big)$, где T_1, ..., T_m — деревья данного леса.) В противном случае установить P равным его последователю в обратном порядке (в представлении (9) это значит, что просто P $\leftarrow$ P $+ c$) и вернуться к шагу F2. ∎

Справедливость алгоритма F доказывается методом индукции по размеру обрабатываемого дерева (см. упр. 16). Этот алгоритм удивительно похож на процедуру

дифференцирования из предыдущего раздела (алгоритм 2.3.2D), которая вычисляет функцию почти такого же типа (см. упр. 3). Та же идея используется во многих программах-интерпретаторах в связи с оценкой арифметических выражений, заданных в постфиксной системе обозначений. Обсуждение этого вопроса будет продолжено в главе 8 (см. также упр. 17, в котором приводится другая важная процедура, подобная алгоритму F).

Таким образом, деревья и леса могут иметь несколько различных представлений последовательного типа. Существует также несколько представлений *связанных* типов, которые описываются ниже.

Первая идея связана с преобразованием представления (3) в представление (6): удалим поля INFO из каждого неконцевого узла и преобразуем эту информацию в новый концевой узел предыдущего узла. Например, деревья (2) в результате такого преобразования примут вид



. (10)

При этом предполагается (без потери общности), что все поля INFO в древовидной структуре находятся в концевых узлах. Следовательно, в естественном представлении бинарного дерева из раздела 2.3.2 поля LLINK и INFO являются взаимно исключающими и могут совместно использовать одно и то же поле в каждом узле. Узел может включать такие поля:

LTAG	LLINK или INFO	RLINK

.

Здесь LTAG указывает, является ли второе поле связью. (Сравните это представление, например, с форматом на основе двух слов (10) из раздела 2.3.2.) Сокращая длину поля INFO с 6 до 3 байт, получим, что каждый узел можно разместить в одном слове. Обратите, однако, внимание на то, что вместо 10 узлов мы теперь используем 15 узлов. Для леса (10) потребуется 15 слов памяти, тогда как для (2) — 20, однако в последнем случае поля INFO занимают 60 байт по сравнению с 30 байт в первом случае. Таким образом, при использовании представления (10) нельзя получить никакой экономии памяти, если не применяется дополнительное пространство в полях INFO. Дело в том, что удаление полей LLINK в (10) компенсируется добавлением почти такого же количества новых полей RLINK в добавленных узлах. Более подробно различия между этими представлениями описываются в упр. 4.

В представлении дерева как стандартного бинарного дерева для поля LLINK точнее было бы использовать название LCHILD, поскольку оно направлено от узла-родителя к крайнему слева узлу-ребенку. Этот узел-ребенок обычно является "самым младшим" ребенком дерева, так как легче всего вставить узел слева от семьи, чем справа. Поэтому сокращение LCHILD может трактоваться, как "последний ребенок" или "крайний ребенок".

Во многих приложениях древовидных структур довольно часто требуется обращаться к узлам дерева как по направлению вверх, так и по направлению вниз. Прошитое дерево предоставляет возможность перехода к верхним узлам, хотя и с небольшой скоростью. Однако иногда лучше иметь в каждом узле третью связь PARENT. При ее наличии получим *трижды связанное дерево* (*triply linked tree*), каждый узел которого имеет связи LCHILD, RLINK и PARENT. На рис. 26 показано представление трижды связанного дерева для (2), а пример его использования приводится в разделе 2.4.

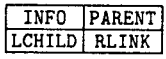
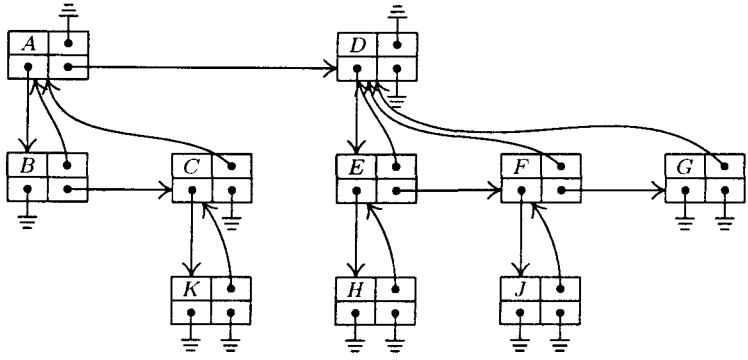


Рис. 26. Трижды связанное дерево



Ясно, что связи PARENT самой по себе достаточно для полного описания структуры любого *ориентированного* дерева (или леса), поскольку схему любого дерева можно нарисовать, зная все направленные вверх связи. Каждый узел, за исключением корня, имеет только одного родителя, но может иметь сразу несколько детей. Поэтому для их определения проще использовать связи, направленные вверх, а не вниз. Почему же тогда они до сих пор не рассматривались? Ответ, конечно же, заключается в том, что направленные вверх связи не всегда пригодны для используемого приложения, так как порой очень трудно быстро определить, является ли узел концевым, или найти какого-либо ребенка узла. Однако есть очень важное приложение, для работы с которым достаточно иметь только связи, направленные вверх. Рассмотрим вкратце элегантный алгоритм обработки отношений эквивалентности, предложенный М. Дж. Фишером (M. J. Fischer) и Б. А. Галлером (B. A. Galler).

Отношением эквивалентности (*equivalence relation*) "$\equiv$" называется отношение между элементами множества S, которое удовлетворяет следующим условиям для любых (необязательно различных) объектов x, y и z множества S.

i) Если $x \equiv y$ и $y \equiv z$, то $x \equiv z$. (Транзитивность.)
ii) Если $x \equiv y$, то $y \equiv x$. (Симметричность.)
iii) $x \equiv x$. (Рефлексивность.)

(Сравните это определение с определением отношения частичного упорядочения, предложенного в разделе 2.2.3. Они сильно различаются несмотря на совпадение двух из трех условий.) Примерами отношений эквивалентности являются отноше-

ния "=", отношение конгруэнтности (по модулю m) для целых чисел, отношение подобия между деревьями, которое определяется в разделе 2.3.1, и т. д.

Задача эквивалентности заключается в считывании нескольких пар эквивалентных элементов и определении с их помощью эквивалентности двух заданных элементов. Например, предположим, что множество S содержит элементы $\{1,2,3,4,5,6,7,8,9\}$ и что даны пары

$$1 \equiv 5, \quad 6 \equiv 8, \quad 7 \equiv 2, \quad 9 \equiv 8, \quad 3 \equiv 7, \quad 4 \equiv 2, \quad 9 \equiv 3. \tag{11}$$

Тогда отсюда следует, например, что $2 \equiv 6$, так как $2 \equiv 7 \equiv 3 \equiv 9 \equiv 8 \equiv 6$. Но при этом нельзя установить, справедливо ли утверждение, что $1 \equiv 6$. Действительно, пары (11) делят множество S на два таких класса

$$\{1,5\} \qquad \text{и} \qquad \{2,3,4,6,7,8,9\}, \tag{12}$$

что два элемента будут эквивалентны тогда и только тогда, когда они принадлежат одному классу. Нетрудно доказать, что *любое* отношение эквивалентности разбивает множество S на такие непересекающиеся *классы эквивалентности* (*equivalence classes*), что два элемента будут эквивалентны тогда и только тогда, когда они принадлежат одному и тому же классу.

Следовательно, решение задачи эквивалентности заключается в определении классов эквивалентности типа (12). Для этого можно начать с ситуации, когда каждый элемент по отдельности образует собственный класс:

$$\{1\}\ \{2\}\ \{3\}\ \{4\}\ \{5\}\ \{6\}\ \{7\}\ \{8\}\ \{9\}. \tag{13}$$

Затем, если дано отношение $1 \equiv 5$, элементы $\{1,5\}$ можно разместить вместе в одном классе. После обработки первых трех отношений $1 \equiv 5$, $6 \equiv 8$ и $7 \equiv 2$ вместо исходного набора классов (13) получим

$$\{1,5\}\ \{2,7\}\ \{3\}\ \{4\}\ \{6,8\}\ \{9\}. \tag{14}$$

На основе отношения $9 \equiv 8$ разместим в одном классе элементы $\{6,8,9\}$ и т. д.

Теперь наша задача состоит в поиске удобного способа представления ситуаций наподобие (12)–(14) внутри компьютера для того, чтобы можно было эффективно выполнять операции слияния классов и проверки принадлежности двух заданных элементов одному классу. Для этого в приведенном ниже алгоритме используются ориентированные древовидные структуры. В нем предполагается, что элементы множества S являются узлами ориентированного леса. Причем два узла эквивалентны вследствие эквивалентности считанных ранее пар *тогда и только тогда, когда они принадлежат одному дереву.* Эту проверку можно довольно просто выполнить, так как два элемента принадлежат одному и тому же дереву тогда и только тогда, когда они являются наследниками одного корня. Более того, слияние двух ориентированных деревьев легко выполняется за счет простого присоединения одного дерева в качестве нового поддерева к корню другого дерева.

Алгоритм E (*Обработка отношений эквивалентности*). Пусть S является множеством чисел $\{1,2,\ldots,n\}$ и пусть `PARENT[1]`, `PARENT[2]`, ..., `PARENT[`n`]` являются целочисленными переменными. В качестве ввода в этом алгоритме используется множество отношений наподобие (11), а для представления множества ориентированных деревьев используется таблица `PARENT`, так что два элемента считаются

эквивалентными вследствие заданных отношений эквивалентности тогда и только тогда, когда они относятся к одному и тому же дереву. (*Замечание.* В более общем случае элементы множества S могут быть представлены в виде символьных имен, а не просто в виде набора чисел от 1 до n. Тогда программа поиска, которая более подробно описывается в главе 6, нашла бы узлы, соответствующие элементам множества S, а элемент таблицы PARENT соответствовал бы полю в этих узлах. Модификация данного алгоритма для такого более общего случая может быть выполнена достаточно просто.)

E1. [Инициализация.] Установить PARENT$[k] \leftarrow 0$ для $1 \leq k \leq n$. (Это значит, что все деревья в исходном состоянии содержат только корень, как в (13).)

E2. [Ввод новой пары.] Получить следующую пару эквивалентных элементов "$j \equiv k$" из входного потока. Если входной поток исчерпан, прекратить выполнение алгоритма.

E3. [Поиск корней.] Если PARENT$[j] > 0$, установить $j \leftarrow$ PARENT$[j]$ и повторить этот шаг. Если PARENT$[k] > 0$, установить $k \leftarrow$ PARENT$[k]$ и повторить этот шаг. (После выполнения операции j и k переходят к корням двух деревьев, которые следует сделать эквивалентными. Отношение на входе $j \equiv k$ избыточно тогда и только тогда, когда $j = k$.)

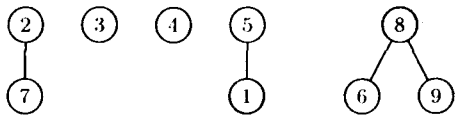
E4. [Слияние деревьев.] Если $j \neq k$, установить PARENT$[j] \leftarrow k$. Вернуться к шагу E2. ▮

Читателю рекомендуется применить этот алгоритм по отношению к входному потоку (11). После обработки отношений $1 \equiv 5$, $6 \equiv 8$, $7 \equiv 2$ и $9 \equiv 8$ получим соответствия

PARENT$[k]$:	5	0	0	0	0	8	2	0	8
k :	1	2	3	4	5	6	7	8	9,

(15)

которые представляют такие деревья:



(16)

С этого момента обработка остальных отношений (11) представляется особенно интересной (см. упр. 9).

На практике задача эквивалентности возникает довольно часто. В разделе 7.4.1 при изучении связности графов будут рассмотрены некоторые существенные усовершенствования алгоритма E. В более общей формулировке эта задача встречается при обработке компилятором "объявлений эквивалентности" в языках наподобие FORTRAN, которая подробно обсуждается в упр. 11.

Помимо перечисленных, существует несколько других способов представления деревьев в памяти компьютера. Напомним три основных метода представления линейных списков, которые рассматривались выше, в разделе 2.2: простейший однонаправленный список с концевой связью Λ, циклически связанные списки и дважды связанные списки. Представление непрошитых бинарных деревьев, рассмотренных в разделе 2.3.1, соответствует простейшему представлению на основе как полей LLINK, так и полей RLINK. Восемь других представлений бинарных деревьев можно получить независимо, используя любой из этих трех методов на основе полей LLINK

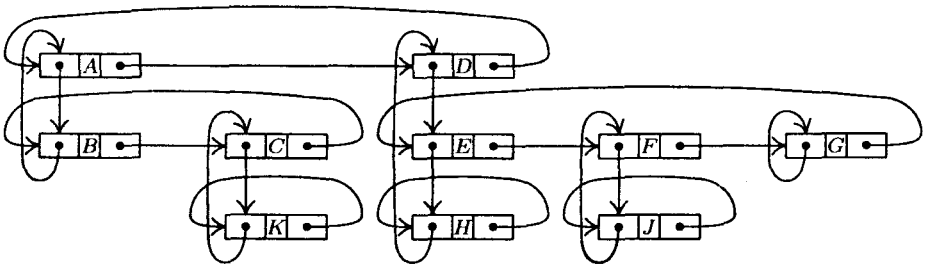


Рис. 27. Кольцевая структура.

и RLINK. Например, на рис. 27 показан результат циклического связывания, которое применяется в обоих направлениях. При использовании циклических связей в обоих направлениях так, как показано на этом рисунке, мы получим *кольцевую структуру* (*ring structure*). Они оказались достаточно гибкими для целого ряда приложений. Оптимальный выбор представления зависит, как обычно, от типов операций вставки, удаления и обхода, которые необходимы в алгоритмах обработки таких структур. Читателю, который внимательно изучил приведенные выше примеры из этой главы, будет нетрудно выбрать и применить на практике какой-либо из вариантов представления.

В заключение раздела приведем пример модифицированной дважды связанной циклической структуры, которая используется для рассмотренной выше задачи, а именно — для полиномиальной арифметики. Алгоритм 2.2.4А выполняет сложение одного полинома с другим при условии, что оба полинома представлены в виде циклических списков. А остальные алгоритмы того же раздела соответствуют другим операциям над полиномами. Полиномы в разделе 2.2.4 содержали не более трех переменных, в то время как при работе с большим количеством переменных удобнее использовать не линейный список, а древовидную структуру.

Полином либо представляет собой константу, либо имеет вид

$$\sum_{0\le j\le n} g_j x^{e_j},$$

где x — переменная, $n > 0$, $0 = e_0 < e_1 < \cdots < e_n$, $g_0, \ldots, g_n$ — полиномы по другим переменным, символьные имена которых в алфавитном порядке располагаются до переменной x, а полиномы $g_1, \ldots, g_n$ отличны от нуля. Это рекурсивное определение полиномов подходит для представления деревьев, как показано на рис. 28. Узлы имеют по шесть полей, которые в случае использования компьютера MIX можно разместить в трех словах:

+	0	LEFT	RIGHT
+	EXP	UP	DOWN
CV			

(17)

Здесь LEFT, RIGHT, UP и DOWN — связи, EXP — целое число, которое указывает степень, а CV — либо константа (коэффициент), либо символьное имя переменной. Для корневого узла UP = Λ, EXP = 0, LEFT = RIGHT = $*$ (связан с самим собой).

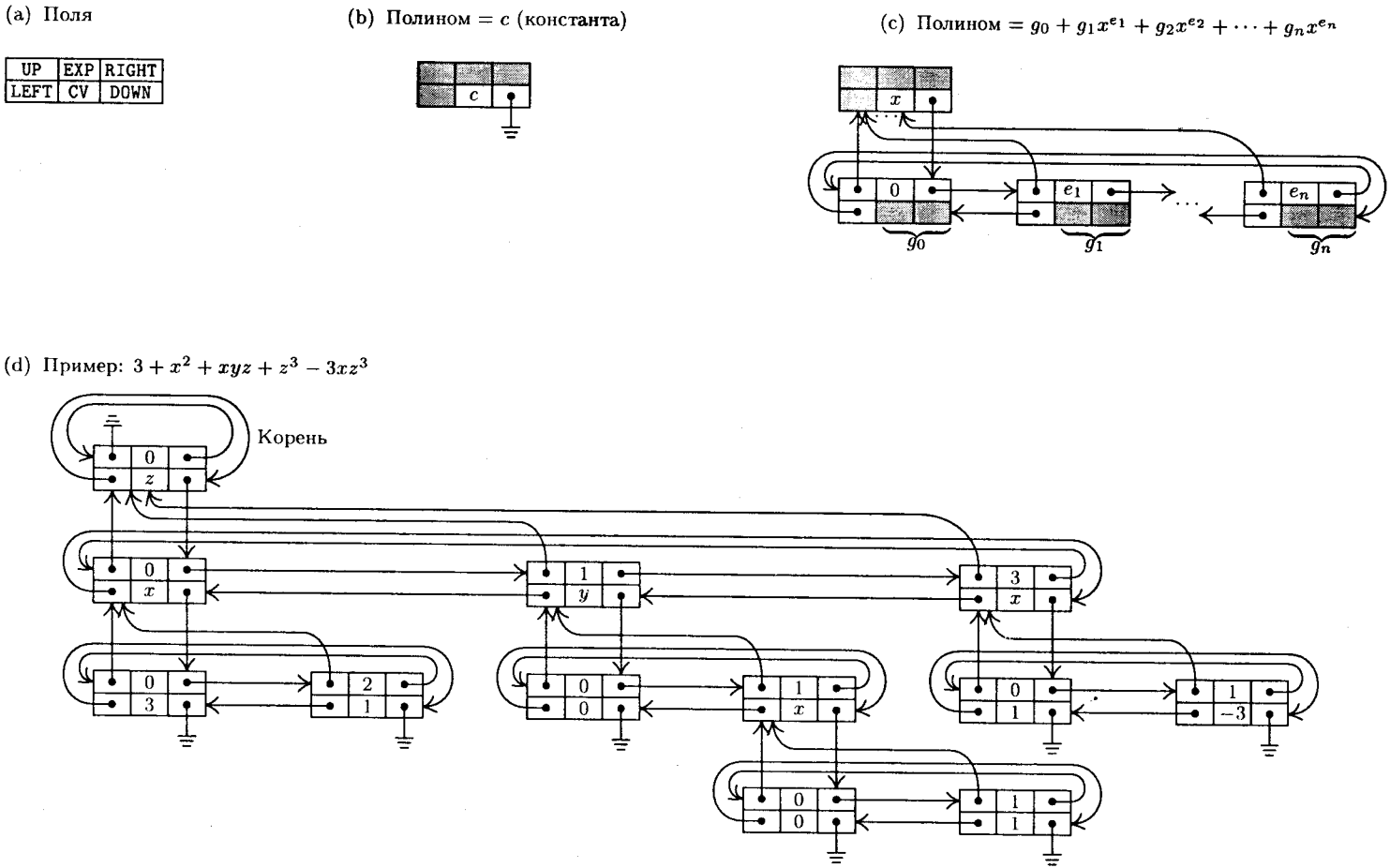


Рис. 28. Представление полиномов на основе связей, направленных в четыре стороны. Заштрихованные поля узлов обозначают данные, которые не имеют никакого отношения к рассматриваемому вопросу.

В следующем алгоритме показано, как в таком дереве со связями по четырем направлениям можно организовать операции обхода, вставки и удаления. А потому он заслуживает особого внимания.

Алгоритм А (*Сложение полиномов*). Этот алгоритм складывает полином (P) с полиномом (Q) при условии, что P и Q являются указательными переменными, которые указывают на корни различных полиномиальных деревьев, подобных показанным на рис. 28. По завершении работы этого алгоритма полином (P) останется в исходном неизменном виде, а полином (Q) будет содержать искомую сумму.

A1. [Проверка типа полинома.] Если DOWN(P) = Λ (т. е. если P указывает на константу), то ни разу не устанавливать или устанавливать Q $\leftarrow$ DOWN(Q) до тех пор, пока не выполнится условие DOWN(Q) = Λ, и перейти к шагу A3. Если DOWN(P) $\neq \Lambda$, то при DOWN(Q) = Λ или CV(Q) < CV(P) перейти к шагу A2. В противном случае, если CV(Q) = CV(P), установить P $\leftarrow$ DOWN(P), Q $\leftarrow$ DOWN(Q) и повторить этот шаг. Если CV(Q) > CV(P), установить Q $\leftarrow$ DOWN(Q) и повторить этот шаг. (На шаге A1 будут найдены два подобных члена в складываемых полиномах или указано на необходимость вставки новой переменной в текущем месте полинома (Q).)

A2. [Вставка по направлению вниз.] Установить R $\Leftarrow$ AVAIL, S $\leftarrow$ DOWN(Q). Если S $\neq \Lambda$, установить UP(S) $\leftarrow$ R, S $\leftarrow$ RIGHT(S) и, если EXP(S) $\neq$ 0, повторять эту операцию до тех пор, пока не получится EXP(S) = 0. Установить UP(R) $\leftarrow$ Q, DOWN(R) $\leftarrow$ DOWN(Q), LEFT(R) $\leftarrow$ R, RIGHT(R) $\leftarrow$ R, CV(R) $\leftarrow$ CV(Q) и EXP(R) $\leftarrow$ 0. Наконец, установить CV(Q) $\leftarrow$ CV(P) и DOWN(Q) $\leftarrow$ R, а затем вернуться к шагу A1. ("Фиктивный" нулевой полином вставляется сразу под узлом NODE(Q), чтобы составить пару с полиномом, найденным внутри дерева P. Обработка связей на этом шаге выполняется очень просто и может быть легко выведена с помощью схемы "до и после", как показано в разделе 2.2.3.)

A3. [Пара найдена.] (Здесь P и Q указывают на соответствующие члены данных полиномов, поэтому можно приступать к сложению.) Установить CV(Q) $\leftarrow$ CV(Q) + CV(P). Если сумма равна нулю и если EXP(Q) $\neq$ 0, перейти к шагу A8. Если EXP(Q) = 0, перейти к шагу A7.

A4. [Продвижение влево.] (После успешного сложения одного члена перейдем к следующему члену, который нужно сложить.) Установить P $\leftarrow$ LEFT(P). Если EXP(P) = 0, перейти к шагу A6. В противном случае ни разу не устанавливать или устанавливать Q $\leftarrow$ LEFT(Q) до тех пор, пока не выполнится условие EXP(Q) $\leq$ EXP(P). Затем, если EXP(Q) = EXP(P), вернуться к шагу A1.

A5. [Вставка справа.] Установить R $\Leftarrow$ AVAIL. Установить UP(R) $\leftarrow$ UP(Q), DOWN(R) $\leftarrow \Lambda$, CV(R) $\leftarrow$ 0, LEFT(R) $\leftarrow$ Q, RIGHT(R) $\leftarrow$ RIGHT(Q), LEFT(RIGHT(R)) $\leftarrow$ R, RIGHT(Q) $\leftarrow$ R, EXP(R) $\leftarrow$ EXP(P) и Q $\leftarrow$ R. Вернуться к шагу A1. (Необходимо вставить новый член в текущей строке справа от узла NODE(Q) в соответствии со степенями текущего члена полинома (P). Как и на шаге A2, эту операцию легче понять, если составить схему "до и после".)

A6. [Возвращение вверх.] (Обход строки полинома (P) полностью завершен.) Установить P $\leftarrow$ UP(P).

A7. [Перевод Q на соответствующий уровень.] Если UP(P) = Λ, перейти к шагу A11. В противном случае ни разу не устанавливать или устанавливать Q ← UP(Q) до тех пор, пока не выполнится условие CV(UP(Q)) = CV(UP(P)). Вернуться к шагу A4.

A8. [Удаление нулевого члена.] Установить R ← Q, Q ← RIGHT(R), S ← LEFT(R), LEFT(Q) ← S, RIGHT(S) ← Q и AVAIL ⇐ R. (Удаление выполнено, поэтому удаляется строка полинома (Q).) Если теперь EXP(LEFT(P)) = 0 и Q = S, перейти к шагу A9; в противном случае вернуться к шагу A4.

A9. [Удаление полинома-константы.] (Аннулирование членов вызвало сокращение полинома до константы, поэтому строка полинома (Q) удаляется.) Установить R ← Q, Q ← UP(Q), DOWN(Q) ← DOWN(R), CV(Q) ← CV(R) и AVAIL ⇐ R. Установить S ← DOWN(Q). Если S ≠ Λ, установить UP(S) ← Q, S ← RIGHT(S) и, если EXP(S) ≠ 0, повторять эту операцию до тех пор, пока не получится EXP(S) = 0.

A10. [Обнаружение нуля?] Если DOWN(Q) = Λ, CV(Q) = 0 и EXP(Q) ≠ 0, установить P ← UP(P) и перейти к шагу A8, в противном случае перейти к шагу A6.

A11. [Прекращение выполнения алгоритма.] Ни разу не устанавливать или устанавливать Q ← UP(Q) до тех пор, пока не получится UP(Q) = Λ (с переводом указателя Q на корень этого дерева). ▮

Данный алгоритм выполняется гораздо быстрее, чем алгоритм 2.2.4A, если полином (P) имеет мало членов, а полином (Q) — много, так как в процессе сложения совершать обход всего полинома (Q) необязательно. Читателю будет очень полезно вручную выполнить алгоритм A, складывая полином $xy - x^2 - xyz - z^3 + 3xz^3$ с полиномом, показанным на рис. 28. (Этот случай не предназначен для демонстрации эффективности данного алгоритма, но позволяет ознакомиться со всеми его шагами и представить все сложные ситуации, которые следует обработать.) Дальнейшее обсуждение алгоритма A приводится в упр. 12 и 13.

Показанное на рис. 28 представление для полиномов от произвольного количества переменных не является "наилучшим". Например, в главе 8 будет рассмотрен другой формат представления полиномов, а также некоторые арифметические алгоритмы на основе вспомогательного стека, которые по сравнению с алгоритмом A обладают существенными преимуществами в отношении концептуальной простоты. Алгоритм A интересен для нас в основном тем, как в нем организована обработка деревьев с большим количеством связей.

УПРАЖНЕНИЯ

▸ **1.** [*20*] Можно ли восстановить связи LLINK, зная только поля LTAG, INFO и RTAG узлов в последовательном порядке уровней, подобном (8)? (Иначе говоря, не являются ли поля LLINK избыточными в представлении (8) и поля RLINK — в представлении (3)?)

2. [*22*] (Задача Беркса, Уоррена и Райта, *Math. Comp.* **8** (1954), 53–57.) Пусть деревья (2) хранятся в памяти в *прямом порядке* с указанием степеней

DEGREE	2	0	1	0	3	1	0	1	0	0
INFO	*A*	*B*	*C*	*K*	*D*	*E*	*H*	*F*	*J*	*G*

[Ср. с (9), где они приведены в обратном порядке.] Создайте алгоритм, аналогичный алгоритму F, для оценки локально определенной функции узлов, выполнив обход этого представления справа налево.

▸ **3.** [*24*] Измените алгоритм 2 3 2D, используя идею алгоритма F промежуточные производные размещались в стеке, а их адреса не записываются таким необычным способом, который применяется на шаге D3 (см упр 2 3 2–21) Управление стеком может осуществляться на основе поля RLINK в корне каждой производной

4. [*18*] Деревья (2) содержат 10 узлов, пять из которых являются концевыми Представление этих деревьев в виде нормальных бинарных деревьев включает 10 полей LLINK и 10 полей RLINK (по одному для каждого узла) Для представления этих деревьев в виде (10), где LLINK и INFO совместно используют одно и то же пространство внутри узла, требуется 5 полей LLINK и 15 полей RLINK В каждом случае имеется по 10 полей INFO

Для леса с n узлами, m из которых являются концевыми, сравните общее количество полей LLINK и RLINK, которые необходимы для каждого из этих двух методов представления дерева

5. [*16*] Трижды связанное дерево, аналогичное показанному на рис 26, содержит в каждом узле поля PARENT, LCHILD и RLINK, причем при отсутствии узла, на который могли бы указать поля PARENT, LCHILD и RLINK, в них применяются связи Λ Стоит ли расширять это представление до *прошитого* дерева, применяя связи-нити вместо пустых связей LCHILD и RLINK, как показано в разделе 2 3 1?

▸ **6.** [*24*] Предположим, что узлы *ориентированного* леса имеют по три поля связи PARENT, LCHILD и RLINK, но только одна связь PARENT используется для обозначения древовидной структуры Поле LCHILD каждого узла равно Λ, а поля RLINK расположены в линейном списке, который просто связывает узлы вместе в некотором порядке Переменная связи FIRST указывает на первый узел, а в последнем узле RLINK $= \Lambda$

Создайте алгоритм обхода этих узлов и заполнения полей LCHILD и RLINK в соответствии со значениями связей PARENT, чтобы в результате можно было получить представление трижды связанного дерева, подобное показанному на рис 26 Кроме того, установите для FIRST значение которое будет указывать на корень первого дерева в этом представлении

7. [*15*] Какие классы содержались бы в (12), если бы в (11) отсутствовало отношение $9 \equiv 3$?

8. [*15*] Алгоритм E задает древовидную структуру, которая представляет заданные пары эквивалентных элементов, но в этом разделе явно не указывается, как можно использовать результат выполнения алгоритма E Создайте алгоритм, который может установить справедливость выражения "$j \equiv k$" при условии, что $1 \le j \le n$, $1 \le k \le n$ и алгоритм E задает таблицу PARENT для некоторого набора пар эквивалентных элементов

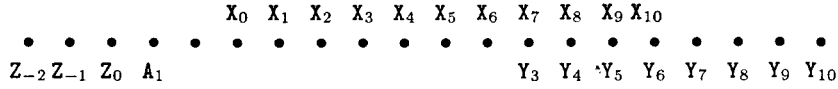
9. [*20*] Предложите таблицу наподобие (15) и схему вида (16), которые изображали бы деревья, полученные после обработки алгоритмом E всех пар эквивалентных элементов в (11) в направлении слева направо

10. [*28*] Для обработки n пар эквивалентных элементов с помощью алгоритма E в худшем случае потребуется выполнить около n^2 шагов Покажите, как можно было бы модифицировать этот алгоритм, чтобы повысить эффективность его работы в данном случае

▸ **11.** [*24*] (*Объявления эквивалентности*) В некоторых компилируемых языках программирования, особенно в языке FORTRAN, предусмотрена возможность перекрытия ячеек памяти, которые выделены для таблиц, последовательно размещенных в памяти Программист предлагает компилятору набор отношений вида "X[j] $\equiv$ Y[k]", который означает, что для переменной X[$j + s$] выделяется та же ячейка памяти, что и для переменной Y[$k + s$] при всех s Кроме того, для каждой переменной определен диапазон допустимых индексов Например, 'ARRAY X[l u]" означает, что нужно выделить некоторую область

памяти для элементов таблицы X[l], X[$l+1$], ..., X[u]. Для каждого класса эквивалентности переменных компилятор резервирует минимально возможный блок последовательно расположенных ячеек памяти, чтобы в нем можно было хранить все элементы таблицы для допустимых значений индексов.

Например, предположим, что имеются таблицы ARRAY X[0:10], ARRAY Y[3:10], ARRAY A[1:1] и ARRAY Z[−2:0], а также пары эквивалентных элементов X[7] ≡ Y[3], Z[0] ≡ A[0] и Y[1] ≡ A[8]. Для этих переменных необходимо выделить 20 последовательных ячеек памяти:



(Адрес расположенной за элементом A[1] ячейки памяти не соответствует ни одному диапазону допустимых значений индексов для любого массива, но эту ячейку все равно придется зарезервировать.)

Назначение этого упражнения заключается в модификации алгоритма E таким образом, чтобы его можно было применять в более общем случае, который только что был описан. Допустим, необходимо создать компилятор такого языка, а таблицы внутри самого компилятора имеют по одному узлу в каждом массиве с полями NAME, PARENT, DELTA, LBD и UBD. Допустим также, что компилятор предварительно обработал все объявления ARRAY таким образом, что при наличии объявления "ARRAY X[l:u]" и P, указывающего на узел X,

$$\texttt{NAME(P)} = \text{“X”}, \quad \texttt{PARENT(P)} = \Lambda, \quad \texttt{DELTA(P)} = 0,$$
$$\texttt{LBD(P)} = l, \quad \texttt{UBD(P)} = u.$$

Задача заключается в создании алгоритма, который обрабатывал бы объявления эквивалентности так, чтобы после выполнения алгоритма получалось следующее:

PARENT(P) = Λ означает, что ячейки памяти X[LBD(P)], ..., X[UBD(P)] должны быть зарезервированы в памяти для этого класса эквивалентности;

PARENT(P) = Q ≠ Λ означает, что ячейка памяти X[k] эквивалентна ячейке Y[k + DELTA(P)], где NAME(Q) = "Y".

Например, до обработки перечисленных выше пар эквивалентных элементов узлы могли выглядеть так.

P	NAME(P)	PARENT(P)	DELTA(P)	LBD(P)	UBD(P)
α	X	Λ	0	0	10
β	Y	Λ	0	3	10
γ	A	Λ	0	1	1
δ	Z	Λ	0	−2	0

А после обработки они могут иметь следующий вид.

α	X	Λ	*	−5	14
β	Y	α	4	*	*
γ	A	δ	0	*	*
δ	Z	α	−3	*	*

(Здесь "*" обозначает данные, которые не имеют никакого отношения к рассматриваемой задаче.)

Создайте алгоритм, который выполняет это преобразование. Предположите, что данные из входного потока поступают в виде (P, j, Q, k), а это означает, что "X[j] ≡ Y[k]", где NAME(P) = "X" и NAME(Q) = "Y". Непременно убедитесь в том, что эти пары эквивалентных

отношений не противоречат одна другой. Например, `X[1]` ≡ `Y[2]` будет противоречить `X[2]` ≡ `Y[1]`.

12. [*21*] В начале алгоритма A переменные `P` и `Q` указывают на корни двух деревьев. Пусть P_0 и Q_0 — значения переменных `P` и `Q` до выполнения алгоритма A. (a) Всегда ли по завершении выполнения этого алгоритма Q_0 будет содержать адрес корня дерева, представляющего результат суммирования двух заданных полиномов? (b) Будут ли переменным `P` и `Q` возвращены их исходные значения P_0 и Q_0 по окончании выполнения этого алгоритма?

▶ **13.** [*M29*] Предложите неформальное доказательство того, что в алгоритме A в начале шага A8 всегда справедливы равенства `EXP(P)` = `EXP(Q)` и `CV(UP(P))` = `CV(UP(Q))`. (Это очень важно для понимания принципа работы алгоритма.)

14. [*40*] Предложите формальное доказательство (или опровержение) справедливости алгоритма A.

15. [*40*] Создайте алгоритм для вычисления произведения двух полиномов, показанных на рис. 28.

16. [*M24*] Докажите корректность алгоритма F.

▶ **17.** [*25*] Алгоритм F позволяет вычислить локально определенную функцию по направлению "снизу вверх", которая сначала вычисляется для детей некоторого узла, а затем — и для самого узла, тогда как локально определенной функцией для узла x по направлению "сверху вниз" f называется функция, которая зависит только от узла x и значения функции f для *родителя* узла x. С помощью вспомогательного стека создайте алгоритм, аналогичный алгоритму F, который оценивает локально определенную функцию по направлению "сверху вниз" f для каждого узла этого дерева. (Подобно алгоритму F данный алгоритм должен эффективно обрабатывать деревья, которые хранятся в *обратном* порядке со степенями узлов, как в (9).)

▶ **18.** [*28*] Создайте алгоритм, который на основе двух таблиц `INFO1[`j`]` и `RLINK[`j`]` для $1 \le j \le n$, соответствующих последовательному представлению в прямом порядке обхода, позволяет создать таблицы `INFO2[`j`]` и `DEGREE[`j`]` для $1 \le j \le n$, соответствующие последовательному представлению в обратном порядке обхода с указанием степеней. Например, согласно (3) и (9) этот алгоритм должен привести таблицы

j	1	2	3	4	5	6	7	8	9	10
`INFO1[`j`]`	*A*	*B*	*C*	*K*	*D*	*E*	*H*	*F*	*J*	*G*
`RLINK[`j`]`	5	3	0	0	0	8	0	10	0	0

к следующему виду:

`INFO2[`j`]`	*B*	*K*	*C*	*A*	*H*	*E*	*J*	*F*	*G*	*D*
`DEGREE[`j`]`	0	0	1	2	0	1	0	1	0	3

19. [*M27*] Вместо использования связей `SCOPE` в (5) можно было бы просто указать количество наследников для каждого узла в прямом порядке:

`DESC`	3	0	1	0	5	1	0	1	0	0
`INFO`	*A*	*B*	*C*	*K*	*D*	*E*	*H*	*F*	*J*	*G*

Пусть $d_1 d_2 \ldots d_n$ — последовательность чисел, указывающих количество наследников для узлов одного леса, полученная таким образом.

a) Покажите, что $k + d_k \le n$ для $1 \le k \le n$ и что из $k \le j \le k + d_k$ следует $j + d_j \le k + d_k$.

b) И наоборот, докажите, что если $d_1 d_2 \ldots d_n$ является последовательностью неотрицательных целых чисел, которые удовлетворяют условиям (a), то она является последовательностью количеств узлов-наследников для данного леса.

c) Предположим, что $d_1 d_2 \ldots d_n$ и $d'_1 d'_2 \ldots d'_n$ — последовательности количеств узлов-наследников для двух лесов. Докажите, что существует третий лес с такой последовательностью количеств узлов-наследников:

$$\min(d_1, d'_1) \min(d_2, d'_2) \ldots \min(d_n, d'_n).$$

2.3.4. Основные математические свойства деревьев

Древовидные структуры еще задолго до изобретения компьютеров были объектом обширных математических исследований, а потому в течение многих лет было накоплено большое количество интересных фактов об их свойствах. В настоящем разделе предложен краткий обзор математической теории деревьев, который позволяет не только глубже понять их природу, но и применить эти результаты на практике в компьютерных алгоритмах.

Читателям, которым не интересна чисто математическая сторона обсуждаемых вопросов, рекомендуется сразу же перейти к подразделу 2.3.4.5, в котором с практической точки зрения рассматривается несколько вопросов, наиболее часто возникающих при использовании описываемых ниже приложений.

Приведенный ниже материал, в основном, относится к большому разделу математики, а именно — к теории графов. К сожалению, в этой области не существует устоявшейся терминологии, и автор придерживается обычной практики при написании современных книг по теории графов. Иначе говоря, здесь используются термины, аналогичные, но не идентичные тем терминам, которые применяются в *других* книгах по теории графов. В следующих подразделах (и далее во всей книге) будет предпринята попытка выбрать короткие, но емкие термины для наиболее важных понятий. Они будут выбраны среди общеупотребительных слов так, чтобы в то же время они не противоречили общепринятой терминологии. Следует учесть, что здесь эта терминология имеет непосредственное отношение к компьютерным приложениям. Поэтому инженер-электрик может назвать деревом то, что здесь называется свободным деревом, так как более краткий термин "дерево" обозначает более широкое понятие, которое часто используется в компьютерной литературе, а потому оно гораздо важнее в компьютерных приложениях. Если следовать терминологии, предлагаемой некоторыми авторами работ по теории графов, то следовало бы вместо термина "дерево" использовать термин "конечное упорядоченное дерево с указанным корнем", а вместо термина "бинарное дерево" — "топологическая раздвоенная древовидность"!

2.3.4.1. Свободные деревья. *Граф* (*graph*) обычно определяется как множество точек (называемых *вершинами* (*vertices*)) вместе с набором линий (называемых *ребрами* (*edges*)), которые соединяют определенные пары вершин. Каждая пара вершин соединяется не более чем одним ребром. Две вершины называются *смежными* (*adjacent*), если их соединяет ребро. Если V и V' являются вершинами и $n \geq 0$, то $(V_0, V_1, \ldots, V_n)$ называется *путем* длины n от вершины V к вершине V', если $V = V_0$, вершина V_k является смежной для вершины V_{k+1} для $0 \leq k < n$, а $V_n = V'$. Путь называется *простым* (*simple*), если различны вершины $V_0, V_1, \ldots, V_{n-1}$ и если различны вершины $V_1, \ldots, V_{n-1}, V_n$. Граф называется *связным* (*connected*), если существует путь между любыми двумя его вершинами. *Циклом* называется простой путь длиной, большей или равной трем, от некоторой вершины к ней самой.

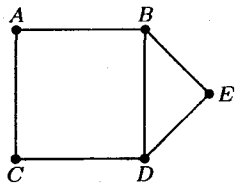


Рис. 29. Граф.

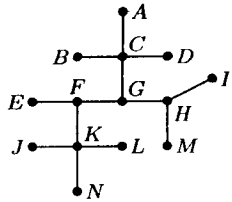


Рис. 30. Свободное дерево.

Эти определения проиллюстрированы на рис. 29, на котором изображен связный граф с пятью вершинами и шестью ребрами. Вершина C является смежной с A, но она не смежная с вершиной B. От вершины B к вершине C есть два пути длиной два: (B, A, C) и (B, D, C). В этом графе имеется несколько циклов, например (B, D, E, B).

Свободное дерево (*free tree*), или дерево без корня (рис. 30), определяется как связный граф без циклов. Это определение применимо как для бесконечных графов, так и для конечных, хотя в компьютерных приложениях обычно используются только конечные деревья. Можно привести несколько эквивалентных способов определения свободного дерева, например некоторые из них представлены в следующей хорошо известной теореме.

Теорема А. *Если G — граф, то для него будут эквивалентными следующие утверждения.*

a) *G — свободное дерево.*

b) *G — связный граф, который при удалении произвольного ребра перестает быть связным.*

c) *Если V и V' — различные вершины графа G, то существует единственный простой путь от вершины V к вершине V'.*

Более того, если граф G конечен и содержит в точности $n > 0$ вершин, следующие утверждения также будут эквивалентны утверждениям (a)–(c).

d) *G не содержит циклов и имеет $n - 1$ ребер.*

e) *G — связный граф, который имеет $n - 1$ ребер.*

Доказательство. Из (a) следует (b), так как при удалении ребра V — V', при котором граф G остается связным, должен существовать простой путь $(V, V_1, \ldots, V')$ длины два или более (см. упр. 2), а тогда $(V, V_1, \ldots, V', V)$ будет циклом в G.

Из (b) следует (c), так как есть по крайней мере один путь от вершины V к вершине V'. И, если бы существовало два таких пути $(V, V_1, \ldots, V')$ и $(V, V_1', \ldots, V')$, можно было бы найти такое наименьшее k, для которого $V_k \neq V_k'$ при удалении ребра V_{k-1} — V_k граф оставался бы связным, так как все еще существует путь $(V_{k-1}, V_k', \ldots, V', \ldots, V_k)$ от вершины V_{k-1} к вершине V_k, который не включает удаленное ребро.

Из (c) следует (a), так как если G содержит цикл $(V, V_1, \ldots, V)$, то существует два простых пути от вершины V к вершине V_1.

Чтобы показать, что (d) и (e) также эквивалентны (a)–(c), сначала докажем вспомогательный результат: в конечном графе G без циклов и хотя бы с одним

ребром существует по крайней мере одна вершина, которая является смежной в точности для одной другой вершины. Докажем это, рассмотрев некоторую вершину V_1 и смежную с ней другую вершину V_2. Для $k \geq 2$ вершина V_k является смежной либо для вершины V_{k-1} и никакой другой, либо для какой-то другой, например с вершиной $V_{k+1} \neq V_{k-1}$. Поскольку в этом графе нет циклов, вершины $V_1, V_2, \ldots, V_{k+1}$ должны быть различными, а потому данный процесс должен в конечном итоге завершиться.

Предположим теперь, что G — это дерево с $n > 1$ вершинами, а V_n — вершина, смежная только для какой-то одной другой вершины, например для вершины V_{n-1}. При удалении вершины V_n и ребра V_{n-1} — V_n полученный в результате граф G' является деревом, так как вершина V_n может находиться в простом пути графа G только в качестве начального или конечного элемента. Таким образом, доказано (методом индукции по n), что G имеет $n-1$ ребер, а значит, из (a) следует (d).

Предположим, что G удовлетворяет условию (d) и величины V_n, V_{n-1} и G' имеют тот же смысл, что и в предыдущем абзаце. Тогда граф G является связным, поскольку вершина V_n связана с вершиной V_{n-1}, которая (по индукции по n) связана со всеми другими вершинами графа G'. Таким образом, из (d) следует (e).

Наконец, предположим, что граф G удовлетворяет условию (e). Если граф G содержит цикл, то можно удалить любое ребро этого цикла, не нарушая связность графа G. Следовательно, продолжая таким образом удалять ребра, получим связный граф G' с $n-1-k$ ребрами и без циклов. Но поскольку из (a) следует (d), то $k = 0$, т. е. $G = G'$. ∎

Понятие свободного дерева можно непосредственно использовать для анализа компьютерных алгоритмов. В разделе 1.3.3 уже рассматривалось применение первого закона Кирхгофа для решения задачи о подсчете числа выполнений каждого шага алгоритма. В результате было найдено, что закон Кирхгофа не позволяет полностью подсчитать это количество, но с его помощью можно сократить число неизвестных, значения которых еще потребуется особым образом интерпретировать. Благодаря теории деревьев можно установить количество оставшихся независимых неизвестных и предложить метод их поиска.

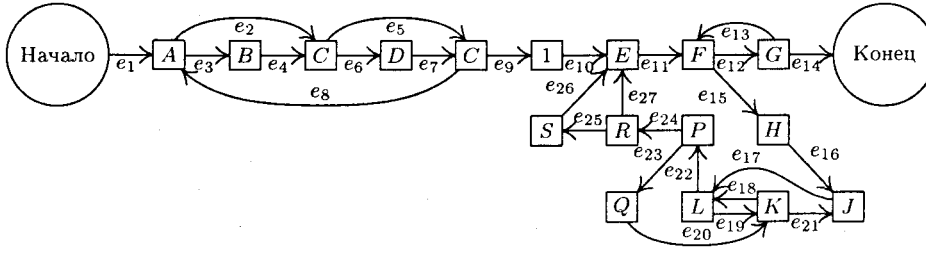


Рис. 31. Абстрактная блок-схема программы 1.3.3A.

Этот метод легче понять на конкретном примере, который впоследствии будет еще не раз использоваться для иллюстрации результатов применения данной теории. На рис. 31 показана абстрактная блок-схема программы 1.3.3A после ее анализа на основе закона Кирхгофа из раздела 1.3.3. Каждый квадратик (т. е. блок) на рис. 31

представляет отдельный шаг вычислений, а буква или число внутри него — количество вычислений, которые выполнялись на этом шаге во время работы программы, согласно обозначениям из раздела 1.3.3. Стрелка между блоками указывает на возможность перехода в программе. Все они отмечены символами $e_1, e_2, \ldots, e_{27}$. Теперь задача заключается в том, чтобы на основе закона Кирхгофа найти все отношения между величинами A, B, C, D, E, F, G, H, J, K, L, P, Q, R и S, а также глубоко проникнуть в суть общей задачи. (*Замечание.* Некоторые упрощения этой задачи уже внесены непосредственно на рис. 31. Например, блок между C и E уже содержит число "1", что является следствием закона Кирхгофа.)

Пусть E_j обозначает количество попыток обхода ветви e_j во время выполнения данной программы. По закону Кирхгофа

$$\begin{gathered}\text{сумма величин } E \text{ на входе в блок} = \text{значение внутри блока} \\ = \text{сумма величин } E \text{ на выходе из блока.}\end{gathered} \tag{1}$$

Например, для блока K получим

$$E_{19} + E_{20} = K = E_{18} + E_{21}. \tag{2}$$

В дальнейшем неизвестными будем считать $E_1, E_2, \ldots, E_{27}$, *а не* $A, B, \ldots, S$.

Блок-схему на рис. 31 можно представить в еще более абстрактном виде, т. е. в виде графа G, как показано на рис. 32. Блоки превратились в вершины, а стрелки $e_1, e_2, \ldots$ теперь представляют собой ребра графа. (Строго говоря, ребра графа не указывают направления, а потому направление стрелок следует игнорировать при рассмотрении теоретических свойств графа G. Однако, как будет показано ниже, стрелки понадобятся при использовании закона Кирхгофа.) Дополнительно ребро e_0, которое проходит от вершины "Начало" до вершины "Конец", вводится для удобства, чтобы закон Кирхгофа был одинаково применим для всех частей графа. В схеме на рис. 32 также внесено несколько изменений по сравнению с блок-схемой, показанной на рис. 31. Дополнительная вершина и ребро добавлены для разбиения стрелки e_{13} на две, e'_{13} и e''_{13}, чтобы соблюдалось основное определение графа (две вершины могут соединяться только одним ребром). Такому же разбиению подверглась и стрелка e_{19}. Аналогичную модификацию схемы следовало бы сделать также для любой вершины со стрелкой, указывающей на эту же вершину.

Некоторые ребра, представленные на рис. 32, выделены более жирным начертанием по сравнению с остальными. Они образуют *свободное поддерево* данного графа, соединяющее все его вершины. В графе блок-схемы всегда можно найти свободное поддерево, поскольку такие графы должны быть связными и согласно п. (b) теоремы A, если связный граф G не является свободным деревом, в нем можно удалить ребро и получить граф без потери связности. Этот процесс можно повторять до тех пор, пока не будет получено искомое поддерево. Другой алгоритм поиска свободного поддерева рассматривается в упр. 6. В любом случае прежде всего следует устранить ребро e_0 (которое проходит от вершины "Начало" до вершины "Конец"). Таким образом, можно предположить, что e_0 в выбранном поддереве отсутствует.

Пусть G' — свободное поддерево графа G, найденное таким образом. Рассмотрим произвольное ребро V — V' графа G, которое *отсутствует* в графе G'. Тогда можно отметить важное следствие из теоремы A: граф G' и его новое ребро V — V' содержат цикл. Действительно, имеется *в точности один* цикл вида $(V, V', \ldots, V)$,

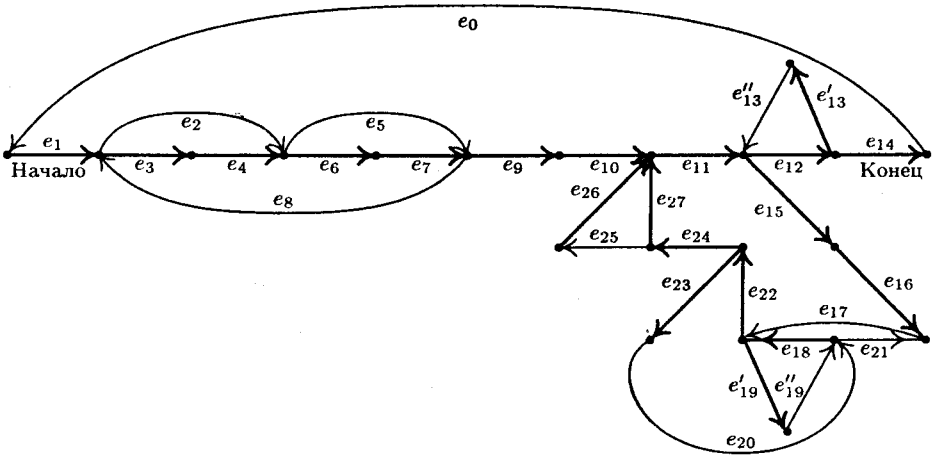


Рис. 32. Граф со свободным поддеревом, соответствующий блок-схеме на рис. 31.

поскольку существует уникальный простой путь от вершины V' до вершины V в графе G'. Например, если G' является свободным деревом, показанным на рис. 32, то, добавив ребро e_2, получим цикл, который сначала проходит через ребро e_2, а затем (в противоположном направлении по отношению к указанным стрелкам) через ребра e_4 и e_3. Этот цикл можно записать в алгебраическом виде "$e_2 - e_4 - e_3$", используя знаки "плюс" и "минус" для обозначения направления обхода, когда он совпадает или не совпадает с направлением стрелок.

Если выполнить этот процесс для каждого ребра, которое не входит в свободное дерево, получатся так называемые *фундаментальные циклы* (*fundamental cycles*), которые для схемы, показанной на рис. 32, выглядят так:

$$
\begin{aligned}
&C_0\colon\ e_0 + e_1 + e_3 + e_4 + e_6 + e_7 + e_9 + e_{10} + e_{11} + e_{12} + e_{14},\\
&C_2\colon\ e_2 - e_4 - e_3,\\
&C_5\colon\ e_5 - e_7 - e_6,\\
&C_8\colon\ e_8 + e_3 + e_4 + e_6 + e_7,\\
&C''_{13}\colon\ e''_{13} + e_{12} + e'_{13},\\
&C_{17}\colon\ e_{17} + e_{22} + e_{24} + e_{27} + e_{11} + e_{15} + e_{16},\\
&C''_{19}\colon\ e''_{19} + e_{18} + e'_{19},\\
&C_{20}\colon\ e_{20} + e_{18} + e_{22} + e_{23},\\
&C_{21}\colon\ e_{21} - e_{16} - e_{15} - e_{11} - e_{27} - e_{24} - e_{22} - e_{18},\\
&C_{25}\colon\ e_{25} + e_{26} - e_{27}.
\end{aligned}
\tag{3}
$$

Очевидно, что ребро e_j, которое не входит в свободное поддерево, будет представлено только в фундаментальных циклах, а именно — в циклах C_j.

Теперь мы вплотную приблизились к кульминационному моменту этого построения. Каждый фундаментальный цикл представляет решение уравнений Кирхгофа. Например, решение, соответствующее циклу C_2, выглядит как $E_2 = +1$, $E_4 = -1$, $E_3 = -1$, а соответствующие всем остальным циклам — как $E = 0$. Ясно, что

коэффициенты вдоль цикла в графе всегда удовлетворяют условию (1) закона Кирхгофа. Более того, уравнения Кирхгофа являются "однородными", так что сумма или разница решений уравнений (1) также является решением. Следовательно, можно сделать вывод, что $E_0, E_2, E_5, \ldots, E_{25}$ являются *независимыми* в следующем смысле.

Если $x_0, x_2, \ldots, x_{25}$ *— произвольные действительные числа* (по одному x_j для каждого ребра e_j, которое не входит в свободное поддерево G'), *то существует такое решение уравнений Кирхгофа* (1), *что* $E_0 = x_0$, $E_2 = x_2$, $\ldots$, $E_{25} = x_{25}$. (4)

Данное решение получено за счет x_0-разового обхода цикла C_0, x_2-разового обхода цикла C_2 и т. д. Более того, значения остальных переменных $E_1, E_3, E_4, \ldots$ полностью *зависят* от значений переменных $E_0, E_2, \ldots, E_{25}$.

Упомянутое в (4) *решение является единственным.* (5)

Если бы существовало два таких решения уравнений Кирхгофа, при которых $E_0 = x_0, \ldots, E_{25} = x_{25}$, можно было бы вычесть одно решение из другого и таким образом получить решение, в котором $E_0 = E_2 = E_5 = \cdots = E_{25} = 0$. Но тогда *все* E_j должны быть равны нулю, так как нетрудно видеть, что нельзя получить ненулевое решение уравнений Кирхгофа для графа, который является свободным деревом (см. упр. 4). Следовательно, два предполагаемых решения должны быть тождественны. Таким образом, доказано, что все решения уравнений Кирхгофа могут быть представлены в виде линейной комбинации решений, полученных на основе фундаментальных циклов.

Применяя эти замечания для графа, показанного на рис. 32, получим следующее общее решение уравнений Кирхгофа на основе независимых переменных $E_0, E_2, \ldots, E_{25}$:

$$
\begin{aligned}
E_1 &= E_0, & E_{14} &= E_0,\\
E_3 &= E_0 - E_2 + E_8, & E_{15} &= E_{17} - E_{21},\\
E_4 &= E_0 - E_2 + E_8, & E_{16} &= E_{17} - E_{21},\\
E_6 &= E_0 - E_5 + E_8, & E_{18} &= E''_{19} + E_{20} - E_{21},\\
E_7 &= E_0 - E_5 + E_8, & E'_{19} &= E''_{19},\\
E_9 &= E_0, & E_{22} &= E_{17} + E_{20} - E_{21},\\
E_{10} &= E_0, & E_{23} &= E_{20},\\
E_{11} &= E_0 + E_{17} - E_{21}, & E_{24} &= E_{17} - E_{21},\\
E_{12} &= E_0 + E''_{13}, & E_{26} &= E_{25},\\
E'_{13} &= E''_{13}, & E_{27} &= E_{17} - E_{21} - E_{25}.
\end{aligned}
\qquad (6)
$$

Чтобы получить эти уравнения, достаточно для каждого ребра e_j поддерева перечислить все такие E_k, для которых ребро e_j входит в цикл C_k, с соответствующим знаком. [Итак, матрица коэффициентов системы уравнений (6) является транспонированной по отношению к матрице коэффициентов системы уравнений (3).]

Строго говоря, цикл C_0 не стоит называть фундаментальным, поскольку он содержит особое ребро e_0. Цикл C_0 без ребра e_0 можно было бы назвать *фундаментальным путем* (*fundamental path*) *от вершины "Начало" до вершины "Конец"*.

При этом граничное условие, которое заключается в том, что блоки "Начало" и "Конец" обрабатываются в точности один раз, эквивалентно отношению

$$E_0 = 1. \tag{7}$$

Выше было показано, как получить все решения с помощью закона Кирхгофа. Этот же метод можно применить не только для блок-схем, но и для анализа электрических цепей (именно так поступил и сам Кирхгоф). Естественно было бы спросить, не являются ли законы Кирхгофа наиболее полным возможным набором уравнений, которые можно было бы предложить для описания блок-схем программ. Иначе говоря, можно ли утверждать, что при каждом выполнении программы от блока "Начало" до блока "Конец" можно получить набор величин $E_1, E_2, \ldots, E_{27}$, которые соответствуют количеству проходов по каждому ребру, причем эти величины подчиняются закону Кирхгофа. Но существуют ли решения уравнений Кирхгофа, которые не отвечают никаким вариантам выполнения компьютерной программы? (Здесь предполагается, что об этой программе ничего, кроме блок-схемы, неизвестно.) Если имеются решения, которые удовлетворяют уравнениям Кирхгофа, но не соответствуют реальному выполнению программы, то можно потребовать выполнения более строгих условий, чем законы Кирхгофа. Для электрических цепей Кирхгоф сформулировал следующий второй закон [*Ann. Physik und Chemie* **64** (1845), 497–514]: сумма падений напряжения в фундаментальном цикле должна быть равна нулю. Но этот закон не применим к данной задаче.

Существует еще одно очевидное условие, которому должны удовлетворять величины E, если они соответствуют некоторому реальному пути в блок-схеме от блока "Начало" до блока "Конец", а именно: они должны быть целыми числами, точнее, *неотрицательными целыми числами*. Это вовсе не тривиальное условие, так как нельзя просто приписать произвольные неотрицательные числа независимым переменным $E_2, E_5, \ldots, E_{25}$. Например, если взять $E_2 = 2$ и $E_8 = 0$, то на основании (6) и (7) получится, что $E_3 = -1$. (Таким образом, нельзя выполнить программу с блок-схемой, представленной на рис. 31, с двойным проходом ребра e_2 без обхода ребра e_8 хотя бы один раз.) Условие неотрицательности значений E не является достаточным. Рассмотрим, например, решение, в котором $E'_{19} = 1$, $E_2 = E_5 = \cdots = E_{17} = E_{20} = E_{21} = E_{25} = 0$. Тогда здесь не существует ни одного пути с проходом по ребру e_{18}, минуя ребро e_{15}. Это необходимое и достаточное условие является ответом на вопрос, поставленный в предыдущем абзаце: для произвольных значений $E_2, E_5, \ldots, E_{25}$ определим $E_1, E_3, \ldots, E_{27}$ согласно (6) и (7). Предположим, что все E — неотрицательные целые числа, а граф с ребрами e_j, для которых $E_j > 0$, и вершинами, которые соединены такими ребрами e_j, является *связным*. Тогда существует путь от блока "Начало" до блока "Конец", в котором ребро e_j проходится в точности E_j раз. Это утверждение доказывается в следующем разделе (см. упр. 2.3.4.2–24).

Подытожим все приведенные выше рассуждения в следующей теореме.

Теорема К. *Если блок-схема (такая, как на рис. 31) содержит n блоков (в том числе блоки "Начало" и "Конец") и m стрелок, то можно найти $m-n+1$ фундаментальных циклов и такой фундаментальный путь от блока "Начало" до блока "Конец", что любой путь от блока "Начало" до блока "Конец" будет эквивалентен (в отношении количества прохождений каждого ребра) одному обходу фундаментального пути и*

единственным образом определенному количеству прохождений каждого фундаментального цикла. (Фундаментальный путь и фундаментальный цикл могут включать несколько ребер, прохождение которых совершается в направлении, *обратном* тому, которое показано стрелкой на этом ребре. В таком случае будем считать, что прохождение ребер осуществляется -1 раз.)

И наоборот, для любого обхода фундаментального пути и фундаментальных циклов, в которых общее количество прохождений каждого ребра неотрицательно и в которых вершины и ребра, соответствующие положительному количеству прохождений, образуют связный граф, существует по крайней мере один эквивалентный путь от блока "Начало" до блока "Конец". ▮

Поиск фундаментальных циклов осуществляется в результате выбора свободного поддерева, аналогичного показанному на рис. 32. Если выбрать другое поддерево, то в общем случае получим другой набор фундаментальных циклов. Существование $m - n + 1$ фундаментальных циклов следует из теоремы А. Причем модификации, которые выполнялись для схемы, показанной на рис. 31, чтобы получить схему на рис. 32, после добавления ребра e_0 не изменяют значения $m - n + 1$, хотя при этом могут возрасти значения m и n. Такое построение можно было бы обобщить с тем, чтобы полностью избавиться от тривиальных модификаций (см. упр. 9).

Теорема К обнадеживает, поскольку в ней говорится, что закон Кирхгофа (который состоит из n уравнений для m неизвестных $E_1, E_2, \ldots, E_m$) обладает лишь одной "избыточной переменной": эти n уравнений позволяют исключить $n-1$ неизвестных. Однако неизвестные переменные в приведенных выше рассуждениях обозначали количество прохождений *ребер*, а не количество входов в каждый *блок* блок-схемы. В упр. 8 показано, как построить другой граф, ребра которого соответствуют блокам блок-схемы, так что описанная выше теория может быть использована для определения истинного числа избыточных переменных.

Способы применения теоремы К в программном обеспечении, используемом для оценки производительности программ на языках программирования высокого уровня, рассмотрены Томасом Боллом (Thomas Ball) и Джеймсом Р. Ларусом (James R. Larus) в работе *ACM Trans. Prog. Languages and Systems* **16** (1994), 1319–1360.

УПРАЖНЕНИЯ

1. [*14*] Перечислите все циклы от вершины B до вершины B, которые содержатся в графе, показанном на рис. 29.

2. [*M20*] Докажите, что если в графе существует путь от вершины V до вершины V', то между этими вершинами также имеется простой путь.

3. [*15*] Какой путь от блока "Начало" до блока "Конец" является эквивалентным (в смысле теоремы К) одному проходу фундаментального пути плюс один проход цикла C_2 на рис. 32?

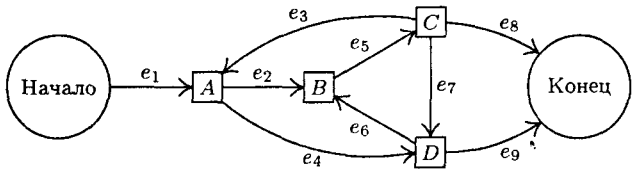
▸ **4.** [*M20*] Пусть G' является конечным свободным деревом, в котором стрелки нарисованы на ребрах $e_1, \ldots, e_{n-1}$. Пусть $E_1, \ldots, E_{n-1}$ — это числа, удовлетворяющие закону Кирхгофа (1) в G'. Покажите, что $E_1 = \cdots = E_{n-1} = 0$.

5. [*20*] Используя уравнения (6), выразите значения $A, B, \ldots, S$, которые находятся внутри блоков на рис. 31, с помощью независимых переменных $E_2, E_5, \ldots, E_{25}$.

▸ **6.** [*M27*] Допустим, что граф содержит n вершин $V_1, \ldots, V_n$ и m ребер $e_1, \ldots, e_m$. Каждое ребро e между вершинами V_a и V_b представлено парой целых чисел (a, b). Создайте

максимально эффективный алгоритм, который использует в качестве входного потока пары чисел $(a_1, b_1), \ldots, (a_m, b_m)$, а на выходе распечатывает подмножество ребер, которые образуют свободное дерево. Если это невозможно, алгоритм должен выдать сообщение об ошибке.

7. [*22*] Выполните описанное в этом разделе построение для блок-схемы

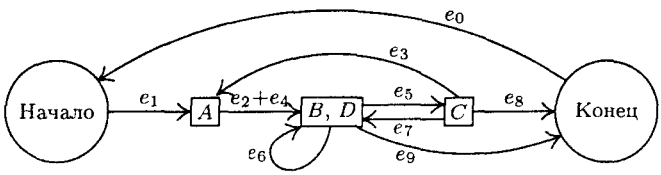


,

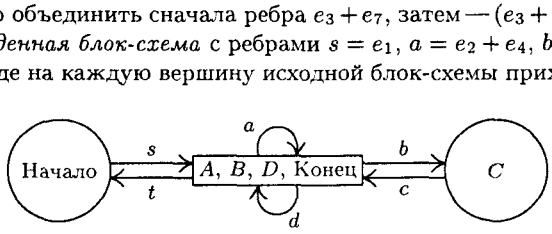
используя для этого свободное поддерево из ребер e_1, e_2, e_3, e_4, e_9. Найдите фундаментальные циклы и выразите E_1, E_2, E_3, E_4, E_9 на основании переменных E_5, E_6, E_7 и E_8.

▶ **8.** [*M25*] Того, кто применяет закон Кирхгофа для программирования блок-схемы, обычно интересуют *потоки через вершины* (*vertex flows*) (т. е. количество прохождений каждого блока для данной блок-схемы), а не потоки через ребра. Например, на схеме в упр. 7 потоки через вершины равны $A = E_2 + E_4$, $B = E_5$, $C = E_3 + E_7 + E_8$, $D = E_6 + E_9$.

Если сгруппировать некоторые вершины, рассматривая их как одну "супервершину", можно объединить потоки ребер, которые соответствуют одному и тому же потоку вершины. Например, в показанной выше блок-схеме ребра e_2 и e_4 можно объединить, если совместить вершины B и D:



(Здесь также от вершины "Начало" до вершины "Конец" проведено ребро e_0.) Продолжая этот процесс, можно объединить сначала ребра $e_3 + e_7$, затем — $(e_3 + e_7) + e_8$ и $e_6 + e_9$, пока не получится *приведенная блок-схема* с ребрами $s = e_1$, $a = e_2 + e_4$, $b = e_5$, $c = e_3 + e_7 + e_8$, $d = e_6 + e_9$, $t = e_0$, где на каждую вершину исходной блок-схемы приходится в точности по одному ребру:



.

По построению в приведенной блок-схеме закон Кирхгофа соблюдается. Новыми потоками ребер здесь являются потоки вершин исходной блок-схемы. Следовательно, применяя упомянутый в этом разделе анализ по отношению к рассматриваемой блок-схеме, можно получить представление о взаимосвязях между исходными потоками вершин.

Докажите, что процесс приведения блок-схемы можно обратить в том смысле, что любое множество потоков $\{a, b, \ldots\}$, которое удовлетворяет закону Кирхгофа в приведенной блок-схеме, может быть "расщеплено" на набор потоков ребер $\{e_0, e_1, \ldots\}$ в исходной блок-схеме. Эти потоки e_j удовлетворяют закону Кирхгофа, и, если их объединить, можно получить потоки $\{a, b, \ldots\}$. Причем некоторые из них могут быть отрицательными. (Хотя здесь показан процесс приведения только для одной частной блок-схемы, данное доказательство должно выполняться в общем случае.)

9. [*M22*] Ребра e_{13} и e_{19}, показанные на рис. 32, расщеплены на две части, поскольку предполагается, что в графе не может быть двух ребер, которые объединяют эти же две вершины. Если взглянуть на окончательный результат построения, то расщепление на две части выглядит достаточно искусственным, потому что наряду с двумя соотношениями $E'_{13} = E''_{13}$ и $E'_{19} = E''_{19}$ в (6) содержатся две независимые переменные: E''_{13} и E''_{19}. Объясните, как это построение можно обобщить, чтобы избежать искусственного расщепления ребер.

10. [*16*] Проектируя электрическую схему компьютера, инженер-электрик приходит к выводу, что необходимо иметь n выводов $T_1, T_2, \ldots, T_n$ с практически одинаковыми значениями рабочего напряжения. Для этого он может спаять провода между любой парой выводов. Смысл этого действия заключается в организации достаточного количества соединений, чтобы существовал путь между любыми двумя выводами. Покажите, что минимальное количество соединений между парами выводов для организации такой сети выводов будет равно $n - 1$, причем $n - 1$ соединений между парами выводов позволяют создать такую сеть тогда и только тогда, когда они образуют свободное дерево (в котором выводы и соединения являются вершинами и ребрами).

11. [*M27*] (R. C. Prim, *Bell System Tech. J.* **36** (1957), 1389–1401.) Рассмотрим задачу о соединениях из упр. 10 с дополнительным условием: для каждой пары $i < j$ задается цена $c(i, j)$, которая обозначает затраты на подключение вывода T_i к выводу T_j. Покажите, что приведенный ниже алгоритм позволяет получить дерево соединений с минимальной ценой. "Если $n = 1$, ничего делать не нужно. В противном случае перенумеровать выводы $\{1, \ldots, n-1\}$ и цены так, чтобы $c(n-1, n) = \min_{1 \le i < n} c(i, n)$; соединить выводы T_{n-1} и T_n; заменить цену $c(j, n-1)$ ценой $\min(c(j, n-1), c(j, n))$ для $1 \le j < n-1$ и повторить этот алгоритм для $n-1$ выводов $T_1, \ldots, T_{n-1}$, используя новые цены. (Алгоритм следует повторять, принимая во внимание то, что каждый раз, когда необходимо создать соединение между выводами T_j и T_{n-1}, соединение на самом деле задается между перенумерованными выводами T_j и T_n, если такое соединение оказывается более дешевым. Таким образом, выводы T_{n-1} и T_n в остальной части алгоритма рассматриваются как один вывод.)" Этот алгоритм можно сформулировать и так: "Сначала следует выбрать какой-то один вывод, затем создавать его самое дешевое соединение с другим выводом до тех пор, пока не будут выбраны все выводы".

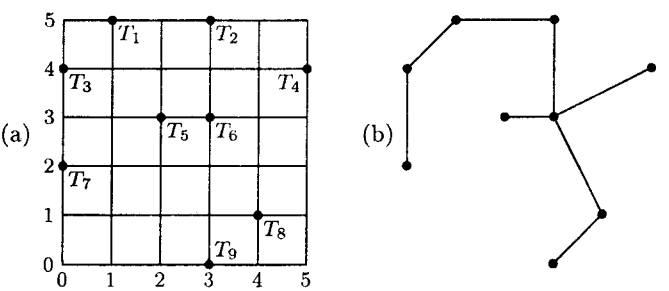


Рис. 33. Свободное дерево с минимальной ценой.

Рассмотрим, например, рис. 33, (a), на котором показана некоторая сетка с девятью выводами. Пусть цена соединения двух выводов определяется его длиной, а именно — расстоянием между выводами. (Читатель может попытаться вручную найти дерево с минимальной ценой, используя интуицию вместо предложенного алгоритма.) Этот алгоритм соединит сначала выводы T_8 и T_9, затем — T_6 и T_8, T_5 и T_6, T_2 и T_6, T_1 и T_2, T_3 и T_1, T_7 и T_3 и, наконец, T_4 соединит либо с T_2, либо с T_6. Дерево с минимальной ценой (с длиной провода $7 + 2\sqrt{2} + 2\sqrt{5}$) показано на рис. 33, (b).

▸ **12.** [*29*] Алгоритм в упр. 11 сформулирован в форме, которая не совсем пригодна для его реализации в компьютерной программе. Перефразируйте его с более подробным описанием всех операций таким образом, чтобы можно было создать компьютерную программу, которая достаточно эффективно их бы выполняла.

13. [*M24*] Рассмотрим граф с n вершинами и m ребрами согласно обозначениям из упр. 6. Покажите, что любую перестановку целых чисел $\{1, 2, \ldots, n\}$ можно представить в виде произведения транспозиций $(a_{k_1} b_{k_1})(a_{k_2} b_{k_2}) \ldots (a_{k_t} b_{k_t})$ тогда и только тогда, когда граф является связным. (Следовательно, существуют множества из $n-1$ транспозиций, которые генерируют все перестановки среди n элементов, но никакое множество из $n-2$ транспозиций не может этого сделать.)

2.3.4.2. Ориентированные деревья. В предыдущем разделе было показано, что абстрактная блок-схема может рассматриваться как граф, если игнорировать направления стрелок на ребрах. Причем употребляемые в теории графов понятия цикла, свободного дерева и другие могут использоваться для изучения блок-схем. Еще больше можно сказать, если учесть направление каждого ребра, так как в этом случае получится "ориентированный граф".

Формально определим *ориентированный граф* (*directed graph* или *digraph*) как множество вершин и множество *дуг* (*arcs*), каждая из которых проходит от вершины V до вершины V'. Если e является дугой от вершины V до вершины V', назовем V *начальной* (*initial*) вершиной дуги e, а V' — *конечной* (*final*) вершиной и запишем $V = \text{init}(e)$, $V' = \text{fin}(e)$. При этом возможен случай, когда $\text{init}(e) = \text{fin}(e)$ (хотя при определении ребра обычного графа он исключается) и несколько различных дуг могут иметь одинаковые начальные и конечные вершины. *Степенью выхода* (*out-degree*) вершины V является количество дуг, которые выходят из нее, а именно — число таких дуг e, что $\text{init}(e) = V$. Аналогично *степень входа* (*in-degree*) вершины V определяется как количество дуг, для которых $\text{fin}(e) = V$.

Хотя понятия пути и цикла для ориентированных графов определяются так же, как и для обычных графов, все же следует рассмотреть некоторые важные новые особенности. Если $e_1, e_2, \ldots, e_n$ являются дугами (с $n \geq 1$), то будем считать, что $(e_1, e_2, \ldots, e_n)$ является *ориентированным путем* (*oriented path*) длины n от вершины V до вершины V', если $V = \text{init}(e_1)$, $V' = \text{fin}(e_n)$, а $\text{fin}(e_k) = \text{init}(e_{k+1})$ для $1 \leq k < n$. Ориентированный путь $(e_1, e_2, \ldots, e_n)$ называется *простым* (*simple*), если $\text{init}(e_1)$, ..., $\text{init}(e_n)$ различны и $\text{fin}(e_1)$, ..., $\text{fin}(e_n)$ различны. *Ориентированный цикл* (*oriented cycle*) — это простой ориентированный путь от некоторой вершины до нее самой. (Ориентированный цикл может иметь длину 1 или 2, хотя такие короткие циклы были исключены из определения цикла в предыдущем разделе. Может ли читатель объяснить, зачем это было нужно?)

Для демонстрации данных определений рассмотрим рис. 31 из предыдущего раздела. Блок с ярлыком "J" является вершиной со степенью входа 2 (в нее входят две дуги e_{16}, e_{21}) и степенью выхода 1. Последовательность $(e_{17}, e_{19}, e_{18}, e_{22})$ является ориентированным путем длины 4 от вершины J до вершины P. Однако этот путь не является простым, например, потому что $\text{init}(e_{19}) = L = \text{init}(e_{22})$. Такая схема не содержит ни одного ориентированного цикла длины 1, но (e_{18}, e_{19}) является ориентированным циклом длины 2.

Ориентированный граф называется *строго связным* (*strongly connected*), если существует ориентированный путь от вершины V до вершины V' для любых двух

вершин $V \neq V'$. Он является *корневым* (*rooted*), если существует хотя бы один *корень* (*root*), т. е. по крайней мере одна такая вершина R, при наличии которой существует ориентированный путь от V к R для всех $V \neq R$. "Строго связный" граф всегда является "корневым", но обратное утверждение не верно. Блок-схема, показанная на рис. 31 в предыдущем разделе, является примером корневого диграфа (т. е. корневого ориентированного графа), корень R которого соответствует вершине-блоку "Начало". Причем, добавляя дугу от блока "Конец" до блока "Начало" (см. рис. 32), получим строго связный граф.

Каждый ориентированный граф G, очевидно, соответствует обыкновенному графу G_0, если игнорировать ориентации и исключить двойные ребра или циклы. Формально выражаясь, граф G_0 содержит ребро от вершины V до вершины V' тогда и только тогда, когда $V \neq V'$ и граф G имеет дугу от вершины V до вершины V' или от вершины V' до вершины V. Рассматривая (неориентированные) *пути* и *циклы* в графе G, мы подразумеваем, что они являются путями и циклами графа G_0. При этом граф G назовем *связным*, если соответствующий граф G_0 является связным (т. е. рассматриваемое свойство не только "слабее" свойства "строго связный", но и еще слабее, чем свойство "корневой").

Ориентированное дерево (*oriented tree*) (рис. 34), иногда называемое некоторыми авторами корневым деревом, представляет собой ориентированный граф с такой вершиной R, что:

a) каждая вершина $V \neq R$ является начальной вершиной в точности одной дуги, которая обозначается как $e[V]$;

b) R не является начальной вершиной ни одной из дуг;

c) R является корнем в указанном выше смысле (т. е. для каждой вершины $V \neq R$ существует ориентированный путь от V к R).

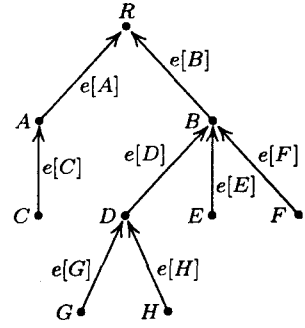


Рис. 34. Ориентированное дерево.

Отсюда немедленно следует, что для каждой вершины $V \neq R$ существует *единственный* ориентированный путь от V к R, а значит, ориентированных циклов не существует.

Легко видеть, что предыдущее определение ориентированного дерева (приведенное в начале раздела 2.3), не противоречит новому определению только в том случае, когда имеется конечное множество вершин. При этом вершины отвечают узлам, а дуга $e[V]$ — это связь между V и `PARENT[V]`.

Соответствующий ориентированному дереву (неориентированный) граф является связным вследствие свойства (c). Более того, он не имеет циклов. Действительно, если $(V_0, V_1, \ldots, V_n)$ является неориентированным циклом с $n \geq 3$ и если $e[V_1]$ — это ребро между V_0 и V_1, то $e[V_2]$ — это ребро между V_1 и V_2 и аналогично $e[V_k]$ — это ребро между V_{k-1} и V_k для $1 \leq k \leq n$, что противоречит отсутствию ориентированных циклов. Если ребро между V_0 и V_1 не равно $e[V_1]$, то оно должно быть равно $e[V_0]$. Тот же аргумент относится к циклу

$$(V_1, V_0, V_{n-1}, \ldots, V_1),$$

так как $V_n = V_0$. Значит, *ориентированное дерево является свободным деревом, если не учитывать направления дуг.*

Следует отметить, что этот процесс можно обратить. Если начать с такого непустого свободного дерева, как на рис. 30, то можно в качестве корня R выбрать *любую* вершину и задать направления для ребер. Если представить себе, что граф "подвесили" за вершину R и встряхнули, то стрелки ребер в нем будут направлены вверх. Более строгая формулировка выглядит так.

> Заменить ребро V — V' дугой от V к V' тогда и только тогда, когда простой путь от V к R проходит через V', т. е. когда он имеет вид $(V_0, V_1, \ldots, V_n)$, где $n > 0$, $V_0 = V$, $V_1 = V'$, $V_n = R$.

Для проверки справедливости этого построения нужно доказать, что для каждого ребра V — V' указано направление $V \leftarrow V'$ (или $V \rightarrow V'$). И это действительно легко сделать, если $(V, V_1, \ldots, R)$ и $(V', V_1', \ldots, R)$ являются простыми путями, т. е. цикл существует всегда, за исключением случаев, когда $V = V_1'$ или $V_1 = V'$. Такое построение демонстрирует, что направления дуг в ориентированном дереве полностью определяются расположением корня, а потому их можно не указывать на схемах, на которых корень обозначен явным образом.

Таким образом, установлена связь между тремя типами деревьев: (упорядоченным) деревом, которое имеет принципиальное значение в компьютерных программах, как показано в начале раздела 2.3, ориентированным (или неупорядоченным) и свободным деревом. Два последних типа деревьев также встречаются при изучении компьютерных алгоритмов, но не так часто, как первый. *Основное различие между этими типами древовидных структур заключается только в объеме структурной информации, которая считается существенной.* Например, на рис. 35 показаны три дерева, которые различны, только если рассматривать их как упорядоченные деревья (с корнем вверху). А если их рассматривать как ориентированные деревья, то идентичными являются первые два, поскольку порядок поддеревьев "слева направо" здесь не существен. Наконец, если считать деревья свободными, то все они на рис. 35 идентичны, так как корень не определен.

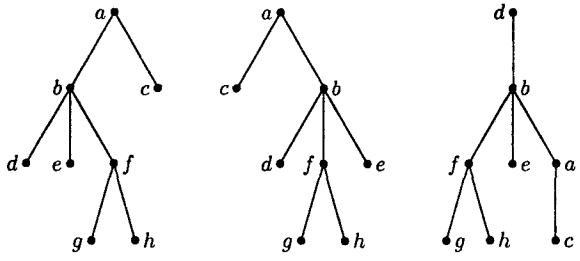


Рис. 35. Три древовидные структуры.

Цепью Эйлера (*Eulerian circuit*) в ориентированном графе является такой ориентированный путь $(e_1, e_2, \ldots, e_m)$, что *каждая дуга* ориентированного графа встречается в этом пути только один раз и $\mathrm{fin}(e_m) = \mathrm{init}(e_1)$. Она представляет собой "полный обход" дуг диграфа. (Цепь Эйлера названа в честь Леонарда Эйлера (Leonhard Euler), который в 1736 году рассмотрел знаменитую задачу о том, что

невозможно обойти во время воскресной прогулки семь мостов Кенигсберга, посетив каждый из них в точности один раз. Он также рассмотрел аналогичную задачу для неориентированных графов. Цепи Эйлера не следует путать с цепями Гамильтона (Hamiltonian circuits), т. е. ориентированными циклами, в которых каждая *вершина* встречается только один раз; см. гл. 7.)

Ориентированный граф называется *сбалансированным* (*balanced*) (рис. 36), если каждая вершина V имеет равные по величине степени входа и выхода, т. е. сколько существует ребер, для которых вершина V является начальной, столько же существует ребер, для которых вершина V является конечной. Это условие тесно связано с законом Кирхгофа (см. упр. 24). Если ориентированный граф имеет цепь Эйлера, то очевидно, что он должен быть связным и сбалансированным, за исключением случаев, когда он имеет *изолированные вершины* (*isolated vertices*), т. е. вершины с равными нулю степенями входа и выхода.

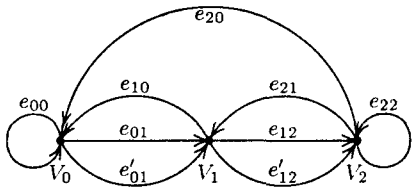


Рис. 36. Сбалансированный ориентированный граф.

Итак, в настоящем разделе дано довольно много определений (ориентированный граф, дуга, начальная вершина, конечная вершина, степень выхода, степень входа, ориентированный путь, простой ориентированный путь, ориентированный цикл, ориентированное дерево, цепь Эйлера, изолированная вершина, а также строгая связность, наличие корня и сбалансированность), но приведено лишь несколько связанных с ними результатов. Теперь мы готовы приступить к изучению более сложного материала. Первым основным результатом является теорема И. Дж. Гуда [I. J. Good, *J. London Math. Soc.* **21** (1947), 167–169], который показал, что цепи Эйлера существуют всегда, кроме случаев, когда они очевидно невозможны.

Теорема G. *Конечный ориентированный граф без изолированных вершин содержит цепь Эйлера тогда и только тогда, когда он связанный и сбалансированный.*

Доказательство. Предположим, что граф G сбалансирован и

$$P = (e_1, \ldots, e_m)$$

представляет собой ориентированный путь максимально возможной длины, в котором ни одна дуга не проходится дважды. Тогда, если $V = \text{fin}(e_m)$ и если k — степень выхода вершины V, то путь P должен включать все k дуг e с начальной вершиной $\text{init}(e) = V$. В противном случае можно было бы добавить e и получить более длинный путь. Но если $\text{init}(e_j) = V$ и $j > 1$, то $\text{fin}(e_{j-1}) = V$. Следовательно, так как граф G сбалансирован, получим

$$\text{init}(e_1) = V = \text{fin}(e_m);$$

в противном случае степень входа вершины V должна быть не меньше $k + 1$.

Теперь, выполнив циклическую перестановку в P, получим, что любая дуга e вне этого пути не имеет общих начальных и конечных вершин с любой дугой этого пути. Поэтому, если P не является цепью Эйлера, граф G не является связным. ▮

Между цепями Эйлера и ориентированными деревьями существует следующая важная взаимосвязь.

Лемма E. *Пусть цепь Эйлера $(e_1, \ldots, e_m)$ ориентированного графа G не имеет изолированных вершин и пусть $R = \text{fin}(e_m) = \text{init}(e_1)$. Пусть для каждой вершины $V \neq R$ ребро $e[V]$ является последним выходом из V в этой цепи, т. е.*

$$e[V] = e_j, \qquad \text{если } \text{init}(e_j) = V, \quad \text{и} \quad \text{init}(e_k) \neq V \quad \text{для } j < k \leq m. \tag{1}$$

Тогда вершины графа G с дугами $e[V]$ образуют ориентированное дерево с корнем R.

Доказательство. Свойства (a) и (b) определения ориентированного дерева, очевидно, удовлетворяются. Согласно результату упр. 7 достаточно только показать, что среди $e[V]$ не существует ориентированных циклов. Но доказательство этого можно получить сразу же, так как если $\text{fin}(e[V]) = V' = \text{init}(e[V'])$, где $e[V] = e_j$ и $e[V'] = e_{j'}$, то $j < j'$. ▮

Эту лемму, возможно, будет легче понять, если рассмотреть ее в обратном направлении, т. е. рассмотреть "первые входы" в каждую вершину. Первые входы образуют неупорядоченное дерево, в котором все дуги направлены *от* R. Лемма E имеет следующую удивительную и очень важную обратную формулировку, справедливость которой доказана Т. ван Аардене-Эренфест и Н. Г. де Брейном [T. van Aardenne-Ehrenfest and N. G. de Bruijn, *Simon Stevin* **28** (1951), 203–217].

Теорема D. *Пусть G — конечный, сбалансированный, ориентированный граф, а G' — ориентированное дерево, состоящее из вершин графа G и нескольких дуг графа G. Пусть R — корень дерева G', а $e[V]$ — дуга дерева G' с начальной вершиной V. Пусть e_1 — произвольная дуга графа G с $\text{init}(e_1) = R$. Тогда ориентированный путь $P = (e_1, e_2, \ldots, e_m)$ будет цепью Эйлера, если для него выполняются следующие условия:*

i) *никакая дуга не проходится более одного раза, т. е. $e_j \neq e_k$ для $j \neq k$;*
ii) *$e[V]$ не используется в P, за исключением единственного случая, который удовлетворяет условию* (i)*, т. е. если $e_j = e[V]$ и если e — дуга с $\text{init}(e) = V$, то $e = e_k$ для некоторого $k \leq j$;*
iii) *путь P заканчивается, только если он не может быть продолжен по правилу* (i)*, т. е. если $\text{init}(e) = \text{fin}(e_m)$, то $e = e_k$ для некоторого k.*

Доказательство. Согласно условию (iii) и доказательству теоремы G получим, что $\text{fin}(e_m) = \text{init}(e_1) = R$. Теперь, если e — дуга, которая не входит в состав пути P, допустим, что $V = \text{fin}(e)$. Так как граф G является сбалансированным, значит, V — это начальная вершина некоторой дуги, не входящей в состав пути P, а если $V \neq R$, то согласно условию (ii) получим, что $e[V]$ не входит в состав пути P. Используем теперь те же доводы с $e = e[V]$ и в конечном итоге получим, что R — начальная вершина некоторой дуги, не входящей в состав этого пути, что противоречит условию (iii). ▮

Суть теоремы D заключается в том, что она демонстрирует простой способ построения цепи Эйлера в сбалансированном ориентированном графе для заданного ориентированного поддерева этого графа (см. пример в упр. 14). Действительно, теорема D позволяет подсчитать точное количество цепей Эйлера в ориентированном графе. Этот результат и другие важные следствия идей, изложенных в данном разделе, излагаются в приведенных ниже упражнениях.

УПРАЖНЕНИЯ

1. [*M20*] Докажите, что если V и V' — вершины ориентированного графа и если существует ориентированный путь от V к V', то существует простой ориентированный путь от V к V'.

2. [*15*] Какие из десяти "фундаментальных циклов" (3) из раздела 2.3.4.1 являются *ориентированными* циклами в ориентированном графе на рис. 32 из того же раздела?

3. [*16*] Нарисуйте схемы для ориентированного графа, который является связным, но не корневым.

▶ **4.** [*M20*] Понятие *топологическая сортировка* (*topological sorting*) для любого конечного ориентированного графа G можно определить как такое линейное упорядочение его вершин $V_1V_2\ldots V_n$, в котором $\text{init}(e)$ предшествует $\text{fin}(e)$ для всех ребер e графа G (см. раздел 2.2.3, рис. 6 и 7). Известно, что ее можно выполнить не для всех конечных ориентированных графов. Для каких графов ее можно осуществить? (Для ответа используйте терминологию из этого раздела.)

5. [*M16*] Пусть G — ориентированный граф, который содержит ориентированный путь $(e_1,\ldots,e_n)$ с $\text{fin}(e_n) = \text{init}(e_1)$. Докажите, что G не является ориентированным деревом, используя предложенную в этом разделе терминологию.

6. [*M21*] Справедливо ли следующее утверждение: "Ориентированный граф, который является корневым и не содержит циклов и ориентированных циклов, является ориентированным деревом"?

▶ **7.** [*M22*] Справедливо ли следующее утверждение: "Ориентированный граф, удовлетворяющий условиям (a) и (b) из определения ориентированного дерева и не имеющий ориентированных циклов, является ориентированным деревом"?

8. [*HM40*] Изучите свойства *группы автоморфизмов* (*automorphism groups*) ориентированных деревьев, т. е. групп, состоящих из всех перестановок π вершин и дуг, для которых $\text{init}(e\pi) = \text{init}(e)\pi$, $\text{fin}(e\pi) = \text{fin}(e)\pi$.

9. [*18*] Указывая направления ребер, нарисуйте схему ориентированного дерева, которое соответствует свободному дереву, показанному на рис. 30, где G — это корень.

10. [*22*] Ориентированное дерево с вершинами $V_1,\ldots,V_n$ можно представить в компьютере с помощью таблицы $P[1],\ldots,P[n]$ следующим образом. Если V_j — корень, то $P[j] = 0$; в противном случае $P[j] = k$, если дуга $e[V_j]$ проходит от V_j к V_k. (Таким образом, $P[1],\ldots,P[n]$ — это такая же таблица, как "родительская" таблица, используемая в алгоритме 2.3.3E.)

В настоящем разделе показано, как свободное дерево может быть преобразовано в ориентированное с помощью выбора произвольной вершины в качестве корня. Следовательно, ориентированное дерево с корнем R можно, пренебрегая направлениями дуг, преобразовать в свободное дерево, а затем задать для них новые направления, получив в итоге ориентированное дерево с корнем в некоторой произвольно выбранной вершиной. Создайте алгоритм, который выполняет такое преобразование заданной таблицы $P[1],\ldots,P[n]$, представляющей ориентированное дерево, в результате которого таблица P

будет представлять это же свободное дерево, но с корнем в вершине V_j, где j — заранее заданное целое число, $1 \le j \le n$.

▸ **11.** [*28*] Используя условия упр. 2.3.4.1-6, но с учетом того, что (a_k, b_k) — это дуга от V_{a_k} к V_{b_k}, создайте алгоритм, который распечатывал бы содержимое не только свободного поддерева, но и фундаментальных циклов. [*Указание.* Можно использовать алгоритм из ответа к упр. 2.3.4.1-6 в сочетании с алгоритмом из предыдущего упражнения.]

12. [*M10*] В соответствии с предложенным здесь определением ориентированного дерева и его определением, данным в начале раздела 2.3, можно ли отождествить *степень* узла дерева со *степенью входа* или *степенью выхода* соответствующей вершины?

▸ **13.** [*M24*] Докажите, что если R — корень, возможно, бесконечного ориентированного графа G, то G содержит ориентированное поддерево с теми же вершинами и корнем R. (Отсюда следует, что в блок-схемах, аналогичных блок-схеме на рис. 32 из раздела 2.3.4.1, всегда можно выбрать свободное поддерево, которое действительно является *ориентированным*. Именно такое поддерево было бы показано на этой схеме, если бы мы выбрали e''_{13}, e''_{19}, e_{20} и e_{17} вместо e'_{13}, e'_{19}, e_{23} и e_{15}.)

14. [*21*] Пусть G — сбалансированный диграф, показанный на рис. 36, а G' — ориентированное поддерево с вершинами V_0, V_1, V_2 и дугами e_{01}, e_{21}. Найдите, начиная с дуги e_{12}, все пути P, которые удовлетворяют условиям теоремы D.

15. [*M20*] Справедливо ли следующее выражение: "Ориентированный, связный и сбалансированный граф является строго связным"?

▸ **16.** [*M24*] В популярном пасьянсе "Часы" обычная колода из 52 карт располагается лицевой стороной вниз в 13 стопках по четыре карты в каждой; причем 12 стопок располагаются по кругу, а стопка 13 — в центре, что, в общем, напоминает циферблат часов. Затем пасьянс раскладывается следующим образом: карта в центральной стопке переворачивается и, если ее значение равно k, размещается возле стопки k. (Значения $1, 2, \ldots, 13$ соответствуют тузу, 2, ..., 10, валету, даме, королю.) Раскладывание продолжается таким образом: верхняя карта из стопки k переворачивается и располагается рядом с ее стопкой и т. д. до тех пор, пока не будет достигнут момент, когда продолжать игру уже невозможно, так как в очередной указанной стопке больше нет карт, которые можно было бы перевернуть. (Игрок не обладает правом выбора варианта продолжения игры, так как эти правила полностью диктуют ход игры.) Игра считается выигранной, если к этому моменту все карты повернуты лицевой стороной вверх. [*См.* E. D. Cheney, *Patience* (Boston: Lee & Shepard, 1870), 62–65; как говорится в книге M. Whitmore Jones, *Games of Patience* (London: L. Upcott Gill, 1900), Chapter 7, в Англии этот пасьянс называется "Пасьянсом путешественника".]

Покажите, что игра выиграна тогда и только тогда, когда следующий ориентированный граф является ориентированным деревом: $V_1, V_2, \ldots, V_{13}$ — вершины, $e_1, e_2, \ldots, e_{12}$ — дуги, где e_j проходит от V_j к V_k, если k — *самая нижняя* карта в стопке j после сдачи карт.

(В частности, если самой нижней картой в стопке j является карта "j" для $j \neq 13$, то легко видеть, что игра, определенно, будет проигрышной, так как эта карта никогда не сможет быть повернута лицевой стороной вверх. Доказанный в настоящем упражнении результат позволяет гораздо быстрее раскладывать такой пасьянс!)

17. [*M32*] Какова вероятность выигрыша при раскладывании пасьянса "Часы" (который описан в упр. 16) при условии, что колода тщательно перетасована? Какова вероятность того, что в точности k карт остаются повернутыми лицевой стороной вниз в момент прекращения игры?

18. [*M30*] Пусть G — граф с $n+1$ вершинами $V_0, V_1, \ldots, V_n$ и m ребрами $e_1, \ldots, e_m$. Преобразуем граф G в ориентированный граф, задав произвольное направление для каждого

ребра, а затем построим матрицу A размера $m \times (n+1)$

$$a_{ij} = \begin{cases} +1, & \text{если } \operatorname{init}(e_i) = V_j; \\ -1, & \text{если } \operatorname{fin}(e_i) = V_j; \\ 0 & \text{в противном случае.} \end{cases}$$

Пусть A_0 — матрица A размера $m \times n$ с удаленным 0-м столбцом.

a) При $m = n$ покажите, что детерминант матрицы A_0 равен 0, если G не является свободным деревом, и равен ± 1, если G *является* свободным деревом.

b) Для произвольного m покажите, что детерминант матрицы $A_0^T A_0$ равен числу свободных поддеревьев графа G (а именно, количеству вариантов выбора n ребер из m ребер таким образом, чтобы полученный граф был свободным деревом). [*Указание.* Используйте условие (a) и результат упр. 1.2.3–46.]

19. [*M31*] (*Теорема о матрице, соответствующей дереву.*) Пусть G — ориентированный граф с $n+1$ вершинами $V_0, V_1, \ldots, V_n$. Пусть A — матрица размера $(n+1) \times (n+1)$ с элементами

$$a_{ij} = \begin{cases} -k, & \text{если } i \neq j \text{ и существует } k \text{ дуг от } V_i \text{ к } V_j; \\ t, & \text{если } i = j \text{ и существует } t \text{ дуг от } V_j \text{ к другим вершинам.} \end{cases}$$

(Отсюда следует, что $a_{i0} + a_{i1} + \cdots + a_{in} = 0$ для $0 \le i \le n$.) Пусть A_0 — это та же матрица, в которой удалены 0-я строка и 0-й столбец. Например, если G является ориентированным графом, который показан на рис. 36, получим

$$A = \begin{pmatrix} 2 & -2 & 0 \\ -1 & 3 & -2 \\ -1 & -1 & 2 \end{pmatrix}, \qquad A_0 = \begin{pmatrix} 3 & -2 \\ -1 & 2 \end{pmatrix}.$$

a) Покажите, что если $a_{00} = 0$ и $a_{jj} = 1$ для $1 \le j \le n$ и если G не имеет дуг, начинающихся и заканчивающихся в одной и той же вершине, то $\det A_0 = [G$ — ориентированное дерево с корнем $V_0]$.

b) Покажите, что в общем случае $\det A_0$ равно количеству ориентированных поддеревьев графа G с корнем V_0 (т. е. количеству способов выбора n дуг из всех дуг графа G, поэтому полученный в результате ориентированный граф является ориентированным деревом с корнем V_0). [*Указание.* Используйте метод индукции по количеству дуг.]

20. [*M21*] Если G — неориентированный граф с $n+1$ вершинами $V_0, \ldots, V_n$, пусть B — матрица размера $n \times n$, которая для $1 \le i, j \le n$ определяется следующим образом:

$$b_{ij} = \begin{cases} t, & \text{если } i = j \text{ и есть } t \text{ ребер, которые соприкасаются с вершиной } V_j; \\ -1, & \text{если } i \neq j \text{ и вершина } V_i \text{ смежна с вершиной } V_j; \\ 0 & \text{в противном случае.} \end{cases}$$

Например, если G — граф, показанный на рис. 29, с $(V_0, V_1, V_2, V_3, V_4) = (A, B, C, D, E)$, то получим

$$B = \begin{pmatrix} 3 & 0 & -1 & -1 \\ 0 & 2 & -1 & 0 \\ -1 & -1 & 3 & -1 \\ -1 & 0 & -1 & 2 \end{pmatrix}.$$

Покажите, что количество свободных поддеревьев графа G равно $\det B$. [*Указание.* Используйте упр. 18 или 19.]

21. [*HM38*] (Задача Т. ван Аардене-Эренфест и Н. Г. де Брейна.) На рис. 36 приведен пример ориентированного графа, который является не только сбалансированным, но и *регулярным* (*regular*). Это означает, что все вершины имеют одинаковые степени входа и

выхода. Пусть G — регулярный диграф с $n+1$ вершинами $V_0, V_1, \ldots, V_n$, каждая вершина которого имеет степень входа и выхода, равную m. (Следовательно, в общем, существует $(n+1)m$ дуг.) Пусть G^* — граф с $(n+1)m$ вершинами, которые соответствуют дугам графа G, и пусть V_{jk} — вершина графа G^*, соответствующая дуге от V_j к V_k в графе G. Дуга проходит от V_{jk} к $V_{j'k'}$ в графе G^* тогда и только тогда, когда $k = j'$. Например, если G — ориентированный граф, показанный на рис. 36, то граф G^* представлен на рис. 37. Цепь Эйлера в графе G является цепью Гамильтона в графе G^*, и наоборот.

Докажите, что количество ориентированных поддеревьев графа G^* в $m^{(n+1)(m-1)}$ раз больше количества ориентированных поддеревьев графа G. [*Указание.* Используйте результат упр. 19.]

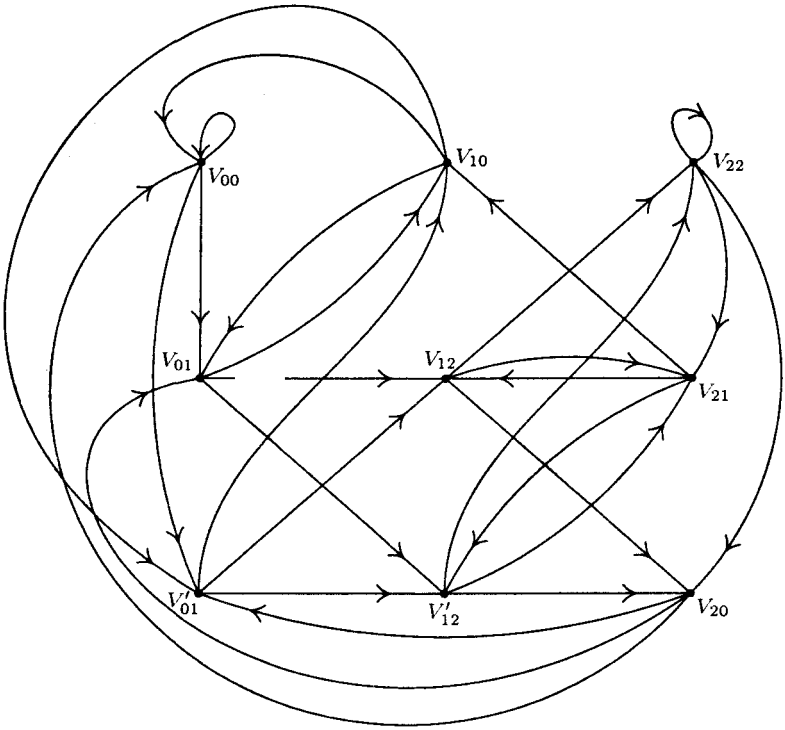


Рис. 37. Дираф с дугами, который соответствует рис. 36 (см. упр. 21).

▸ **22.** [*M26*] Пусть G — сбалансированный, ориентированный граф с вершинами V_1, V_2, ..., V_n без изолированных вершин. Пусть σ_j равно степени выхода вершины V_j. Покажите, что количество цепей Эйлера для графа G равно

$$(\sigma_1 + \sigma_2 + \cdots + \sigma_n)\, T \prod_{j=1}^{n} (\sigma_j - 1)!,$$

где T — количество ориентированных поддеревьев графа G с корнем V_1. *Замечание.* Множитель $(\sigma_1 + \cdots + \sigma_n)$, который равен количеству дуг графа G, можно опустить, если цепь Эйлера $(e_1, \ldots, e_m)$ считается равной $(e_k, \ldots, e_m, e_1, \ldots, e_{k-1})$.]

▸ **23.** [*M33*] (Задача Н. Г. де Брейна.) Для каждой последовательности неотрицательных целых чисел $x_1, \ldots, x_k$, меньших, чем m, допустим, что $f(x_1, \ldots, x_k)$ — неотрицательное

целое число, меньшее, чем m. Определим бесконечную последовательность таким образом: $X_1 = X_2 = \cdots = X_k = 0$; $X_{n+k+1} = f(X_{n+k}, \ldots, X_{n+1})$, где $n \geq 0$. Для какого количества из этих m^{m^k} возможных функций f последовательность будет периодичной с максимальным периодом m^k? [*Указание.* Постройте ориентированный граф с вершинами $(x_1, \ldots, x_{k-1})$ для всех $0 \leq x_j < m$ и с дугами от $(x_1, x_2, \ldots, x_{k-1})$ к $(x_2, \ldots, x_{k-1}, x_k)$; примените результаты упр. 21 и 22.]

▸ **24.** [*M20*] Пусть G — связный дираф с дугами $e_0, e_1, \ldots, e_m$. Пусть $E_0, E_1, \ldots, E_m$ — множество положительных целых чисел, которые удовлетворяют закону Кирхгофа для графа G, т. е. для каждой вершины V,

$$\sum_{\text{init}(e_j)=V} E_j = \sum_{\text{fin}(e_j)=V} E_j.$$

Также предположим, что $E_0 = 1$. Докажите, что в графе G существует такой ориентированный путь от $\text{fin}(e_0)$ к $\text{init}(e_0)$, что ребро e_j содержится в нем E_j раз для $1 \leq j \leq m$, тогда как ребро e_0 не входит в него вообще. [*Указание.* Примените теорему G к соответствующему ориентированному графу.]

25. [*26*] Создайте компьютерное представление ориентированных графов, которые обобщают представление дерева в виде правопрошитого бинарного дерева. Используйте два поля связи ALINK, BLINK и два однобитовых поля ATAG, BTAG так, чтобы это представление обладало следующими свойствами: (i) для каждой *дуги* (а *не* каждой вершины) ориентированного графа существует один узел; (ii) если ориентированный граф является ориентированным деревом с корнем R и если добавить дугу от R к новой вершине H, то представление ориентированного графа будет точно таким же, как представление этого ориентированного дерева в виде правопрошитого бинарного дерева (с некоторым порядком, накладываемым на детей каждой семьи) в том смысле, что поля ALINK, BLINK, BTAG соответствуют полям LLINK, RLINK, RTAG из раздела 2.3 2; (iii) представление является симметричным в том смысле, что обмен полей ALINK и ATAG с BLINK и BTAG эквивалентен изменению направления всех дуг ориентированного графа.

▸ **26.** [*HM39*] (*Анализ случайного алгоритма.*) Пусть G — ориентированный граф с вершинами $V_1, V_2, \ldots, V_n$. Допустим, что G представляет блок-схему алгоритма, где V_1 — вершина блока "Начало" и V_n — вершина блока "Конец". (Следовательно, вершина V_n — корень графа G.) Предположим, что каждой дуге e графа G приписана вероятность $p(e)$, которая удовлетворяет условиям

$$0 < p(e) \leq 1; \qquad \sum_{\text{init}(e)=V_j} p(e) = 1 \quad \text{для } 1 \leq j < n.$$

Рассмотрим случайный путь, который начинается в вершине V_1, а дуги e графа G последовательно выбираются с вероятностью $p(e)$ до тех пор, пока не будет достигнута вершина V_n; причем выбор дуги на каждом этапе не зависит от сделанных ранее выборов.

Рассмотрим, например, граф из упр. 2.3.4.1–7 и присвоим вероятности 1, $\frac{1}{2}$, $\frac{1}{2}$, $\frac{1}{2}$, 1, $\frac{3}{4}$, $\frac{1}{4}$, $\frac{1}{4}$, $\frac{1}{4}$ дугам $e_1, e_2, \ldots, e_9$. Тогда путь "Начало–*A*–*B*–*C*–*A*–*D*–*B*–*C*–Конец" будет выбран с вероятностью $1 \cdot \frac{1}{2} \cdot 1 \cdot \frac{1}{2} \cdot \frac{1}{2} \cdot \frac{3}{4} \cdot 1 \cdot \frac{1}{4} = \frac{3}{128}$.

Такие случайные пути называются *цепями Маркова* (*Markov chains*) в честь русского математика Андрея Андреевича Маркова, который первым провел интенсивные исследования подобных стохастических процессов. Эту ситуацию можно применять для моделирования некоторых алгоритмов, хотя используемое условие выбора каждой дуги независимо от предыдущего пути является чрезвычайно сильным предположением. Назначение данного упражнения заключается в анализе времени вычисления алгоритмов такого типа.

Этот анализ можно упростить, если рассмотреть матрицу $A = (a_{ij})$ размера $n \times n$, где $a_{ij} = \sum p(e)$ и сумма вычисляется по всем таким дугам e, которые проходят от вершины V_i к вершине V_j. Если таких дуг нет, то $a_{ij} = 0$. В рассмотренном выше примере матрица A выглядит так:

$$\begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{2} & 0 & \frac{1}{2} & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & \frac{1}{2} & 0 & 0 & \frac{1}{4} & \frac{1}{4} \\ 0 & 0 & \frac{3}{4} & 0 & 0 & \frac{1}{4} \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}.$$

Отсюда сразу же следует, что $(A^k)_{ij}$ — это вероятность того, что путь, который начинается в вершине V_i, закончится в вершине V_j после k шагов.

Докажите справедливость приведенных ниже утверждений для произвольного ориентированного графа G указанного типа.

a) Матрица $(I - A)$ не является сингулярной. [*Указание.* Покажите, что не существует такого ненулевого вектора x, для которого $xA^n = x$.]

b) Среднее количество появлений вершины V_j в этом пути равно

$$(I - A)^{-1}_{1j} = \text{cofactor}_{j1}(I - A)/\det(I - A) \qquad \text{для } 1 \le j \le n,$$

где $\text{cofactor}_{lk}(I-A)$ — принятое здесь и далее обозначение алгебраического дополнения элемента в l-й строке и k-м столбце матрицы $(I-A)$. [Таким образом, в рассмотренном примере показано, что вершины A, B, C, D в среднем проходятся $\frac{13}{6}$, $\frac{7}{3}$, $\frac{7}{3}$, $\frac{5}{3}$ раз.]

c) Вероятность того, что вершина V_j встречается в этом пути, равна

$$a_j = \text{cofactor}_{j1}(I - A)/\text{cofactor}_{jj}(I - A);$$

причем $a_n = 1$, а потому данный путь с вероятностью "единица" прекращается спустя конечное количество шагов.

d) Вероятность того, что случайный путь, начинаясь в вершине V_j, никогда вновь не пройдет через вершину V_j, равна $b_j = \det(I - A)/\text{cofactor}_{jj}(I - A)$.

e) Вероятность того, что вершина V_j входит в точности k раз в этот путь, равна $a_j(1 - b_j)^{k-1}b_j$ для $k \ge 1$, $1 \le j \le n$.

27. [*M30*] (*Устойчивые состояния.*) Пусть G — ориентированный граф с вершинами $V_1, \ldots, V_n$, дуги которого имеют вероятности $p(e)$ согласно определениям из упр. 26. Однако предположим, что вместо указания вершин "Начало" и "Конец" используется условие строгой связности графа G. Таким образом, каждая вершина V_j является корнем, а вероятности $p(e)$ положительны и удовлетворяют условию $\sum_{\text{init}(e)=V_j} p(e) = 1$ для всех j. Тогда случайный процесс, подобный описанному в упр. 26, называется устойчивым состоянием ("steady state") $(x_1, \ldots, x_n)$, если

$$x_j = \sum_{\text{fin}(e)=V_j} p(e)\, x_{\text{init}(e)}, \qquad 1 \le j \le n.$$

Пусть t_j — это сумма произведений $\prod_{e \in T_j} p(e)$, взятая по всем ориентированным поддеревьям T_j графа G с корнем в вершине V_j. Докажите, что $(t_1, \ldots, t_n)$ является устойчивым состоянием случайного процесса.

▸ **28.** [*M35*] Рассмотрим детерминант матрицы размера $(m+n)\times(m+n)$, который показан ниже, для случая, когда $m=2$ и $n=3$:

$$\det\begin{pmatrix} a_{10}+a_{11}+a_{12}+a_{13} & 0 & a_{11} & a_{12} & a_{13} \\ 0 & a_{20}+a_{21}+a_{22}+a_{23} & a_{21} & a_{22} & a_{23} \\ b_{11} & b_{12} & b_{10}+b_{11}+b_{12} & 0 & 0 \\ b_{21} & b_{22} & 0 & b_{20}+b_{21}+b_{22} & 0 \\ b_{31} & b_{32} & 0 & 0 & b_{30}+b_{31}+b_{32} \end{pmatrix}.$$

Покажите, что при его разложении по степеням a и b каждый ненулевой член будет иметь коэффициент $+1$. Сколько членов будет содержать такое разложение? Предложите правило для ориентированных деревьев, которое точно описывает, какие члены будут присутствовать в этом разложении.

***2.3.4.3. Лемма о бесконечном дереве.** До сих пор, в основном, рассматривались деревья только с конечным количеством вершин (узлов), но определения, данные для свободных и ориентированных деревьев, можно также применить для бесконечных графов. Бесконечные *упорядоченные* деревья можно определить несколькими способами. Можно, например, расширить понятие десятичной системы обозначений Дьюи до бесконечных совокупностей чисел так, как это сделано в упр. 2.3–14. Даже при изучении компьютерных алгоритмов иногда возникает необходимость исследовать свойства бесконечных алгоритмов, например, для доказательства от противного того, что некоторое дерево *не является* бесконечным. Одно из наиболее фундаментальных свойств бесконечных деревьев, впервые сформулированное в довольно общей форме Д. Кенигом (D. König), выглядит так.

Теорема К (*Лемма о бесконечном дереве*). *В каждом бесконечном ориентированном дереве, в котором каждая вершина имеет конечную степень, имеется бесконечный путь к корню, т. е. бесконечная последовательность вершин V_0, V_1, V_2, ..., в которой V_0 — корень и $\mathrm{fin}(e[V_{j+1}]) = V_j$ для всех $j \geq 0$.*

Доказательство. Определим этот путь, начиная с вершины V_0, которая является корнем ориентированного дерева. Предположим, что для некоторого $j \geq 0$ выбрана такая вершина V_j, которая имеет бесконечно много наследников. Предполагается, что степень узла V_j конечна, а потому V_j имеет конечное количество детей $U_1, \ldots, U_n$. По крайней мере один такой ребенок должен иметь бесконечное количество наследников. Предположим, например, что вершина V_{j+1} является таким ребенком вершины V_j.

Тогда получим, что V_0, V_1, V_2, ... — это бесконечный путь с началом в указанном корне. ∎

Студенты, изучающие математический анализ, могут легко узнать, что здесь используется такой же аргумент, как и при доказательстве классической теоремы Больцано-Вейерштрасса о том, что "ограниченное бесконечное множество действительных чисел имеет предельную точку". Другая формулировка теоремы К принадлежит Кенигу: "Если человечество никогда не вымрет, то некто из ныне живущих принадлежит роду, который никогда не вымрет".

При первом знакомстве с теоремой К складывается впечатление, что она абсолютно очевидна. Но после более внимательного обдумывания и изучения примеров ее использования становится ясно, что она имеет гораздо более глубокий смысл.

Хотя степень каждого узла данного дерева конечна, не предполагается, что степени *ограничены* (т. е. степень каждой вершины меньше некоторого N). Поэтому могут существовать узлы со все более и более высокими степенями. Теперь, по крайней мере, ясно, что несмотря на то, что, в конце концов, потомки всех современников вымрут, есть семьи, которые будут продолжать свой род в миллионах, миллиардах и т. д. поколений. Действительно, Г. В. Ватсон (H. W. Watson) опубликовал "доказательство" того, что если некоторые законы биологической вероятности выполняются бесконечно долго, то в будущем родится бесконечное количество людей, но каждый род вымрет с вероятностью "единица". Несмотря на небольшую ошибку, которая привела его к такому ложному выводу (интересно отметить, что он не посчитал свои выводы логически противоречивыми), в его статье, *J. Anthropological Inst. Gt. Britain and Ireland* **4** (1874), 138–144, содержатся очень важные и глубокие теоремы.

Противопоставленное для теоремы К утверждение непосредственно применимо для компьютерных алгоритмов: *если некий алгоритм периодически делится на некоторое ограниченное подмножество подалгоритмов, причем каждая цепь подалгоритмов конечна, то будет конечным и сам алгоритм.*

Иначе говоря, пусть существует такое конечное или бесконечное множество S, что каждый его элемент является последовательностью $(x_1, x_2, \ldots, x_n)$ положительных целых чисел с конечной длиной $n \geq 0$. Если выполняются условия

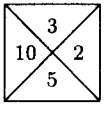
i) если $(x_1, \ldots, x_n)$ принадлежит S, то $(x_1, \ldots, x_k)$ также принадлежит S для $0 \leq k \leq n$;

ii) если $(x_1, \ldots, x_n)$ принадлежит S, то существует только конечное множество таких значений x_{n+1}, для которых $(x_1, \ldots, x_n, x_{n+1})$ также принадлежит S;

iii) не существует такой бесконечной последовательности $(x_1, x_2, \ldots)$, для которой все ее начальные подпоследовательности $(x_1, x_2, \ldots, x_n)$ принадлежат S,

то множество S является, по существу, ориентированным деревом, обозначение которого записано в десятичной системе обозначений Дьюи, и согласно теореме К множество S *конечно.*

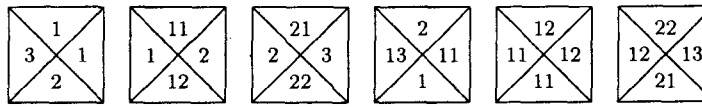
Некоторые наиболее убедительные примеры демонстрации потенциальных возможностей теоремы К относятся к целому ряду интересных задач о покрытиях плоскости, предложенных Хао Вангом (Hao Wang). *Тетрадный тип* (*tetrad type*) — это квадрат (или тетрада), разделенный на четыре части, в каждой из которых указан некоторый номер, например так, как схематически показано ниже:



. (1)

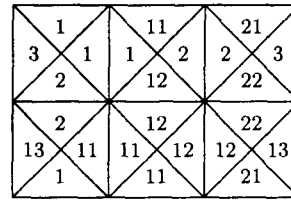
Задача *покрытия плоскости* (*tiling the plane*) заключается в следующем. Пусть имеется некоторое конечное множество тетрадных типов с неограниченным количеством тетрад каждого типа. Требуется предложить способ покрытия бесконечной плоскости тетрадами (без вращения или зеркального отображения тетрадных типов) таким образом, чтобы две тетрады были смежными, только если они соприкасаются сторонами с одинаковыми числами на них. Например, плоскость можно покрыть

шестью тетрадными типами



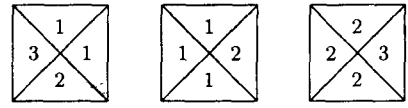
(2)

только одним способом, а именно — повторив прямоугольник



(3)

бесконечное количество раз. Читатель может убедиться в том, что нельзя покрыть плоскость такими тремя тетрадными типами:



(4)

Ванг заметил [*Scientific American* **213**, 5 (November, 1965), 98–106], что *если можно покрыть верхний правый квадрант плоскости, то можно покрыть и всю плоскость.* Это совершенно неожиданный результат, так как в методе покрытия верхнего правого квадранта подразумевается наличие "границы" по осям x и y, причем нет никакого намека на то, как можно покрыть верхний левый квадрант плоскости (поскольку тетрадные типы нельзя вращать или зеркально отображать). При этом нельзя отделаться от границы просто за счет сдвига верхнего правого квадранта вниз и влево, так как сдвиг имеет смысл выполнять только на конечную величину. Вот как выглядит доказательство Ванга. Из существования решения для покрытия правого верхнего квадранта следует, что для любого n существует способ покрытия квадрата $2n \times 2n$. Множество всех решений задачи покрытия квадратов с четной длиной сторон образует ориентированное дерево, если дети каждого решения x размера $2n \times 2n$ являются решениями размера $(2n+2) \times (2n+2)$, которые могут быть получены путем окаймления решения x. Корнем такого дерева является решение размера 0×0, его детьми — решение размера 2×2 и т. д. Каждый узел имеет только ограниченное количество детей, так как в задаче покрытия плоскости предполагается, что для покрытия может использоваться только конечное множество тетрадных типов. Следовательно, согласно лемме о бесконечном дереве существует бесконечный путь к корню. Это значит, что существует способ покрытия всей плоскости (хотя, возможно, его не так уж просто будет найти!).

Описание дальнейшего развития исследований в области тетрадного покрытия плоскости можно найти в прекрасной книге Б. Грюнбаума и Дж. К. Шепарда [*Tilings and Patterns* by B. Grünbaum and G. C. Shephard (Freeman, 1987), Chapter 11].

УПРАЖНЕНИЯ

1. [*M10*] В этом разделе идет речь о множестве S, которое содержит конечные последовательности положительных целых чисел, а также приводится утверждение о том, что

данное множество представляет собой "существенно ориентированное дерево". Что в таком случае является корнем и дугами этого ориентированного дерева?

2. [*20*] Покажите, что если допускается вращение тетрадных типов, то решение существует всегда.

▸ **3.** [*M23*] Если можно покрыть верхний правый квадрант плоскости с помощью заданного *бесконечного* множества тетрадных типов, то всегда ли можно покрыть ими всю плоскость?

4. [*M25*] (Задача Х. Ванга.) Шесть тетрадных типов (2) позволяют получить тороидальное решение этой задачи покрытия плоскости, т. е. решение, в котором некоторый прямоугольный фрагмент, а именно — (3), будет повторяться по всей плоскости.

Предположим без доказательства, что для каждого покрытия плоскости с помощью тетрадных типов существует тороидальное решение, в котором применяются эти тетрадные типы. Используйте данное предположение вместе с леммой о бесконечном дереве для создания алгоритма, с помощью которого для заданных спецификаций любого конечного множества тетрадных типов можно определить за конечное число шагов, существует ли способ покрытия плоскости этими тетрадными типами.

5. [*M40*] Покажите, что, используя 92 показанных ниже тетрадных типа, можно покрыть плоскость, но при этом не существует тороидального решения в смысле, указанном в упр. 4.

Для упрощения спецификации 92 типов прежде всего введем некоторые условные обозначения. Введем такие "основные коды".

$$\begin{array}{llll}
\alpha = (1,\ 2,\ 1,\ 2) & \beta = (3,\ 4,\ 2,\ 1) & \gamma = (2,\ 1,\ 3,\ 4) & \delta = (4,\ 3,\ 4,\ 3) \\
a = (Q, D,\ P, R) & b = (\ ,\ , L,\ P) & c = (U, Q,\ T,\ S) & d = (\ ,\ , S, T) \\
N = (Y,\ \ , X,\ \ ) & J = (D, U,\ \ , X) & K = (\ ,\ Y, R,\ L) & B = (\ ,\ ,\ ,\ ) \\
R = (\ ,\ , R, R) & L = (\ ,\ , L,\ L) & P = (\ ,\ ,\ P,\ P) & S = (\ ,\ , S, S) \\
 & T = (\ ,\ , T,\ T) & X = (\ ,\ , X, X) & \\
Y = (Y,\ Y,\ \ ,\ \ ) & U = (U, U,\ \ ,\ \ ) & D = (D, D,\ \ ,\ \ ) & Q = (Q, Q,\ \ ,\ \ )
\end{array}$$

Тогда тетрадные типы будут иметь следующий вид.

$$\begin{array}{ll}
\alpha\{a, b, c, d\} & \text{[4 типа]} \\
\beta\{Y\{B, U, Q\}\{P, T\},\ \{B, U, D, Q\}\{P, S, T\},\ K\{B, U, Q\}\} & \text{[21 тип]} \\
\gamma\{\{\{X, B\}\{L, P, S, T\}, R\}\{B, Q\},\ J\{L, P, S, T\}\} & \text{[22 типа]} \\
\delta\{X\{L, P, S, T\}\{B, Q\},\ Y\{B, U, Q\}\{P, T\},\ N\{a, b, c, d\}, & \\
\quad J\{L, P, S, T\},\ K\{B, U, Q\},\ \{R, L, P, S, T\}\{B, U, D, Q\}\} & \text{[45 типов]}
\end{array}$$

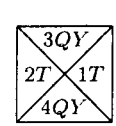
Эти сокращения означают, что основные коды нужно представить покомпонентно во всех указанных сочетаниях и рассортировать по алфавиту внутри каждого компонента. Например, сокращение

$$\beta Y\{B, U, Q\}\{P, T\}$$

обозначает шесть таких тетрадных типов: βYBP, βYUP, βYQP, βYBT, βYUT, βYQT. Тип βYQT после покомпонентного умножения соответствующих друг другу компонентов (где умножение на пропущенный компонент обозначает умножение на единицу) примет вид

$$(3, 4, 2, 1)(Y, Y,\ \ ,\ \ )(Q, Q,\ \ ,\ \ )(\ \ ,\ \ , T, T) = (3QY, 4QY, 2T, 1T).$$

Предполагается, что он соответствует показанному справа тетрадному типу, в четырех частях которого вместо чисел используются символьные строки. При этом два тетрадных типа могут располагаться рядом только при совпадении строк соприкасающихся частей.



β-тетрадой называется тетрадный тип, если в его спецификации указанного выше вида содержится β. Начиная поиск решения этой задачи, обратите внимание, что слева и справа от β-тетрады должны находиться α-тетрады, снизу и сверху — δ-тетрады. Справа от αa-тетрады должны располагаться βKB, βKU или βKQ, за которыми должна следовать αb-тетрада, и т. д.

(Это построение является упрощенной версией построения, предложенного Робертом Бергером (Robert Berger), который доказал, что общая задача из упр. 4 без учета неправильного предположения не может быть разрешена. См. *Memoirs Amer. Math. Soc.* **66** (1966).)

▸ **6.** [*M23*] (Задача Отто Шрайера (Otto Schreier).) В своей знаменитой статье [*Nieuw Archief voor Wiskunde* (2) **15** (1927), 212–216] Б. Л. Ван дер Варден (B. L. van der Waerden) доказал следующую теорему.

> *Если k и m — положительные целые числа и если есть k множеств $S_1, \ldots, S_k$ положительных целых чисел и каждое положительное целое число принадлежит по крайней мере одному из этих множеств, то хотя бы одно из множеств S_j содержит арифметическую прогрессию длины m.*

(Это значит, что существуют такие целые числа a и $\delta > 0$, что $a + \delta$, $a + 2\delta$, $\ldots$, $a + m\delta$ находятся в множестве S_j.) Если возможно, попробуйте использовать этот результат и лемму о бесконечном дереве для доказательства следующего, более "сильного", результата.

> *Если k и m — положительные целые числа, то существует такое N, что если есть k множеств $S_1, \ldots, S_k$ целых чисел, причем каждое значение между 1 и N включено по крайней мере в одно из этих множеств, то хотя бы одно из множеств S_j содержит арифметическую прогрессию длины m.*

Интересное описание Ван дер Вардена о том, как было найдено доказательство данной теоремы, можно найти в книге *Studies in Pure Mathematics*, ed. by L. Mirsky (Academic Press, 1971), 251–260.

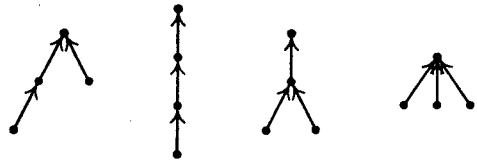
▸ **7.** [*M30*] Если это возможно, используя теорему Ван дер Вардена из упр. 6 и лемму о бесконечном дереве, докажите следующее утверждение.

> *Если k — положительное целое число и имеется k множеств $S_1, \ldots, S_k$ целых чисел, причем каждое положительное целое число включено по крайней мере в одно из этих множеств, то хотя бы одно из множеств S_j содержит бесконечно длинную арифметическую прогрессию.*

▸ **8.** [*M39*] (Задача Дж. Б. Крускала (J. B. Kruskal).) Если T и T' — конечные и упорядоченные деревья, пусть $T \subseteq T'$ обозначает, что T можно вложить в T' так, как в упр. 2.3.2–22. Докажите, что если T_1, T_2, T_3, $\ldots$ — бесконечная последовательность деревьев, то существуют такие целые числа $j < k$, что $T_j \subseteq T_k$. (Иначе говоря, докажите, что можно построить бесконечную последовательность деревьев, в которой ни одно дерево не содержит построенных ранее деревьев этой последовательности. Данный факт можно использовать для доказательства того, что некоторые алгоритмы должны завершиться за конечное число шагов.)

***2.3.4.4. Перечисление деревьев.** Некоторые наиболее поучительные примеры применения математической теории деревьев для анализа алгоритмов связаны с формулами подсчета количества различных деревьев того или иного типа. Например, если поставить вопрос о том, сколько различных ориентированных деревьев можно построить, используя четыре неразличимые вершины, то получится, что

возможны только следующие четыре типа деревьев:



(1)

В первой из рассматриваемых здесь задач перечисления деревьев определим количество a_n структурно различных ориентированных деревьев с n вершинами. Очевидно, что $a_1 = 1$. Если $n > 1$, то дерево имеет корень и поддеревья. Допустим, что существует j_1 поддеревьев с одной вершиной, j_2 — с двумя вершинами и т. д. Тогда можно выбрать j_k деревьев из a_k возможных деревьев с k вершинами

$$\binom{a_k + j_k - 1}{j_k}$$

способами, так как в подобном случае допускаются повторения (см. упр. 1.2.6–60). Исходя из этого, получаем следующее:

$$a_n = \sum_{j_1+2j_2+\cdots=n-1} \binom{a_1 + j_1 - 1}{j_1} \cdots \binom{a_{n-1} + j_{n-1} - 1}{j_{n-1}} \quad \text{для } n > 1. \tag{2}$$

Рассмотрим производящую функцию $A(z) = \sum_n a_n z^n$ с $a_0 = 0$. Находим, что из тождества

$$\frac{1}{(1-z^r)^a} = \sum_j \binom{a+j-1}{j} z^{rj}$$

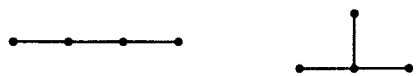
и уравнения (2) следует, что

$$A(z) = \frac{z}{(1-z)^{a_1}(1-z^2)^{a_2}(1-z^3)^{a_3}\dots}. \tag{3}$$

Этим выражением для $A(z)$ не очень удобно пользоваться, так как оно содержит бесконечное произведение и коэффициенты $a_1, a_2, \dots$ в правой части. Более элегантный способ представления $A(z)$ предлагается в упр. 1. Он позволяет получить значительно более эффективную формулу для подсчета значений a_n (см. упр. 2) и фактически может использоваться для описания асимптотического поведения a_n для больших n (см. упр. 4). Таким образом, получаем

$$A(z) = z + z^2 + 2z^3 + 4z^4 + 9z^5 + 20z^6 + 48z^7 + 115z^8 + 286z^9 + 719z^{10} + 1842z^{11} + \cdots. \tag{4}$$

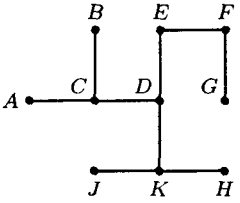
Теперь, когда число ориентированных деревьев найдено, интересно было бы определить количество структурно различных *свободных деревьев* с n вершинами. Существует только два различных свободных дерева с четырьмя вершинами, а именно



(5)

так как два первых и два последних дерева (1) будут тождественны, если не принимать во внимание направление.

Как было показано выше, в свободном дереве в качестве корня можно выбрать любую вершину X и единственным образом задать направления ребер так, что это дерево станет ориентированным с корнем X. Допустим, что для заданной вершины X корень X содержит k поддеревьев с $s_1, s_2, \ldots, s_k$ вершинами в данных поддеревьях. Ясно, что k — это количество дуг, которые касаются вершины X, а $s_1 + s_2 + \cdots + s_k = n - 1$. Тогда *весом* (*weight*) вершины X назовем величину $\max(s_1, s_2, \ldots, s_k)$. Таким образом, вершина D дерева



(6)

имеет вес 3 (каждое из поддеревьев вершины D содержит по три из девяти остальных вершин), а вершина E имеет вес $\max(7, 2) = 7$. Вершина с минимальным весом называется *центроидом* (*centroid*) свободного дерева.

Пусть X и $s_1, s_2, \ldots, s_k$ определены так же, как выше, и пусть $Y_1, Y_2, \ldots, Y_k$ — корни поддеревьев, которые выходят из X. Вес Y_1 должен быть по крайней мере равен $n - s_1 = 1 + s_2 + \cdots + s_k$, поскольку, если предполагается, что Y_1 — корень, в его поддереве существует $n - s_1$ вершин и X в том числе. Если поддерево Y_1 содержит центроид Y, получим

$$\text{высота}\,(X) = \max(s_1, s_2, \ldots, s_k) \geq \text{высота}\,(Y) \geq 1 + s_2 + \cdots + s_k.$$

Это возможно, только если $s_1 > s_2 + \cdots + s_k$. Аналогичный результат можно получить, если вместо Y_1 в данных рассуждениях использовать Y_j. Поэтому *центроид может находиться не более чем в одном поддереве данной вершины.*

Это довольно "сильное" условие, из которого следует, что *в свободном дереве существует не более двух центроидов и, если имеются два центроида, они являются смежными* (см. упр. 9).

И наоборот, если $s_1 > s_2 + \cdots + s_k$, в поддереве Y_1 *имеется* центроид, так как

$$\text{высота}\,(Y_1) \leq \max(s_1 - 1,\ 1 + s_2 + \cdots + s_k) \leq s_1 = \text{высота}\,(X)$$

и вес всех узлов в поддеревьях $Y_2, \ldots, Y_k$ по крайней мере равен $s_1 + 1$. Таким образом доказано, что *вершина X является единственным центроидом свободного дерева тогда и только тогда, когда*

$$s_j \leq s_1 + \cdots + s_k - s_j \qquad \text{для } 1 \leq j \leq k. \tag{7}$$

Следовательно, количество свободных деревьев с n вершинами, имеющих только один центроид, равно количеству ориентированных деревьев с n вершинами минус количество таких ориентированных деревьев, для которых нарушается условие (7). К последним относятся ориентированные деревья с s_j вершинами и ориентированные деревья с $n - s_j \leq s_j$ вершинами. Тогда количество свободных деревьев с одним центроидом равно

$$a_n - a_1 a_{n-1} - a_2 a_{n-2} - \cdots - a_{\lfloor n/2 \rfloor} a_{\lceil n/2 \rceil}. \tag{8}$$

Свободное дерево с двумя центроидами имеет четное количество вершин, а вес каждого центроида равен $n/2$ (см. упр. 10). Поэтому, если $n = 2m$, количество свободных деревьев с двумя центроидами равно количеству вариантов выбора двух объектов из a_m объектов с повторением, а именно:

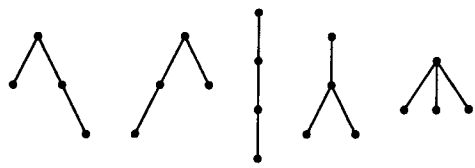
$$\binom{a_m + 1}{2}.$$

Следовательно, чтобы получить общее количество свободных деревьев, сложим $\frac{1}{2}a_{n/2}(a_{n/2}+1)$ с (8) для четных n. Взглянув на выражение (8), можно предложить простую производящую функцию. И действительно, легко получить, что *производящая функция для структурно различных свободных деревьев равна*

$$\begin{aligned} F(z) &= A(z) - \frac{1}{2}A(z)^2 + \frac{1}{2}A(z^2) \\ &= z + z^2 + z^3 + 2z^4 + 3z^5 + 6z^6 + 11z^7 + 23z^8 \\ &\qquad + 47z^9 + 106z^{10} + 235z^{11} + \cdots. \end{aligned} \tag{9}$$

Это простое соотношение между $F(z)$ и $A(z)$ впервые было получено М. Э. К. Джорданом (M. E. C. Jordan), который занимался исследованием данной задачи еще в 1869 году.

Приступим теперь к задаче о перечислении *упорядоченных деревьев*, которые имеют особо существенное значение для программирования компьютерных алгоритмов. На основе четырех вершин можно построить пять следующих структурно различных упорядоченных деревьев:



(10)

Первые два являются идентичными, если рассматривать их как ориентированные деревья, поэтому только одно из них было показано выше, в (1).

Прежде чем перейти к подсчету различных упорядоченных древовидных структур, рассмотрим *бинарные деревья*, так как они ближе к действительному представлению данных внутри компьютера и их проще исследовать. Пусть b_n — количество различных бинарных деревьев с n узлами. Согласно определению бинарных деревьев очевидно, что $b_0 = 1$ и для $n > 0$ количество их различных вариантов равно числу способов расположения бинарного дерева с k узлами слева от корня и другого бинарного дерева с $n - 1 - k$ узлами справа. Поэтому

$$b_n = b_0 b_{n-1} + b_1 b_{n-2} + \cdots + b_{n-1} b_0, \qquad n \geq 1. \tag{11}$$

Из данного соотношения ясно, что производящая функция

$$B(z) = b_0 + b_1 z + b_2 z^2 + \cdots$$

удовлетворяет уравнению

$$zB(z)^2 = B(z) - 1. \tag{12}$$

Решая это квадратное уравнение с учетом того факта, что $B(0) = 1$, получим

$$\begin{aligned} B(z) &= \frac{1}{2z}\left(1-\sqrt{1-4z}\right) = \frac{1}{2z}\left(1-\sum_{k\ge 0}\binom{\frac{1}{2}}{k}(-4z)^k\right) \\ &= 2\sum_{n\ge 0}\binom{\frac{1}{2}}{n+1}(-4z)^n = \sum_{n\ge 0}\binom{-\frac{1}{2}}{n}\frac{(-4z)^n}{n+1} \\ &= \sum_{n\ge 0}\binom{2n}{n}\frac{z^n}{n+1} \\ &= 1+z+2z^2+5z^3+14z^4+42z^5+132z^6+429z^7 \\ &\qquad +1430z^8+4862z^9+16796z^{10}+\cdots. \end{aligned} \tag{13}$$

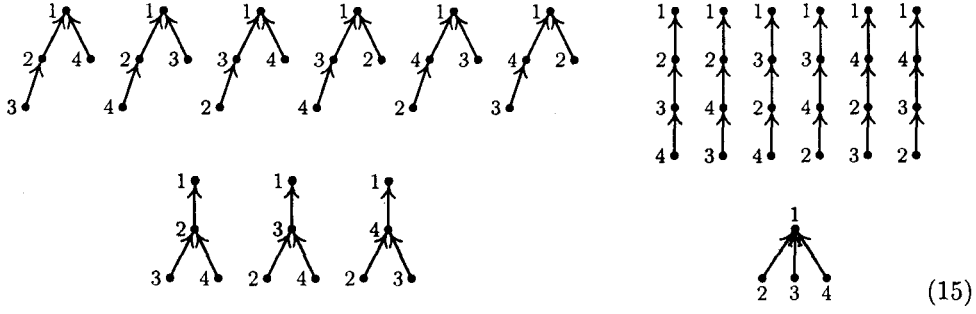
(См. упр. 1.2.6–47.) Следовательно, искомый ответ таков:

$$b_n = \frac{1}{n+1}\binom{2n}{n}. \tag{14}$$

По формуле Стирлинга это асимптотически равно $4^n/n\sqrt{\pi n}+O(4^n n^{-5/2})$. Некоторые важные обобщения уравнения (14) приводятся в упр. 11 и 32.

Возвращаясь к задаче о подсчете упорядоченных деревьев с n узлами, видим, что, по сути, это задача о вычислении количества бинарных деревьев, так как ранее было установлено естественное соответствие между бинарными деревьями и лесами, а дерево минус корень как раз и является лесом. Следовательно, *количество упорядоченных деревьев с n вершинами равно b_{n-1}, т. е. количеству бинарных деревьев с $n-1$ вершинами.*

При выполнении приведенных выше перечислений предполагалось, что вершины являются неразличимыми. Если отметить ярлыками вершины 1–4 из (1) и считать корнем вершину 1, то получится 16 различных ориентированных деревьев.



(15)

Ясно, что задача перечисления помеченных деревьев существенно отличается от описанной выше задачи. В этом случае ее можно перефразировать так: "Рассмотрим три линии, проходящие от вершин 2–4 к другой вершине. Для каждой из них существует три варианта, а в целом имеется $3^3 = 27$ вариантов. Сколько из них соответствует различным ориентированным деревьям с корнем 1?". Как показано выше, таких вариантов 16. Аналогичная формулировка той же задачи для n вершин звучит следующим образом: "Пусть $f(x)$ — такая целочисленная функция, что $f(1) = 1$ и $1 \le f(x) \le n$ для всех целых $1 \le x \le n$. Назовем f *отображением дерева*

(*tree mapping*), если $f^{[n]}(x)$, т. е. выполненная n раз итерация $f(f(\cdots(f(x))\cdots))$ этой функции равна 1 для всех x. Сколько в таком случае существует отображений дерева?". Эта задача возникает, например, в связи с генерированием случайных чисел. К большому удивлению, можно обнаружить, что в среднем точно одна из каждых n таких функций f является отображением дерева.

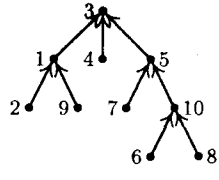
Можно легко решить задачу перечисления с помощью общих формул для подсчета поддеревьев графа, которые получены в предыдущих разделах (см. упр. 12). Однако существует и более информативный способ решения, который позволяет получить новый и компактный метод представления ориентированной древовидной структуры.

Предположим, что дано ориентированное дерево с вершинами $\{1, 2, \ldots, n\}$ и дугами $n-1$, которые проходят от j к $f(j)$ для всех j, за исключением корня. В нем существует по крайней мере одна концевая вершина (лист), поэтому предположим, что V_1 — наименьший номер листа. Если $n > 1$, запишем $f(V_1)$ и удалим из дерева вершину V_1 и дугу $V_1 \to f(V_1)$. Пусть V_2 — наименьший номер листа, который получился в результате предыдущей операции. Если $n > 2$, запишем $f(V_2)$ и удалим вершину V_2 и дугу $V_2 \to f(V_2)$ из дерева, а затем будем продолжать в таком духе до тех пор, пока из данного дерева не будут удалены все вершины, кроме корня. Таким образом получим последовательность из $n-1$ чисел

$$f(V_1),\ f(V_2),\ \ldots,\ f(V_{n-1}), \qquad 1 \le f(V_j) \le n, \tag{16}$$

которая называется ***каноническим представлением*** (*canonical representation*) исходного ориентированного дерева.

Например, ориентированное дерево



(17)

с 10 вершинами имеет такое каноническое представление: 1, 3, 10, 5, 10, 1, 3, 5, 3.

Важной особенностью здесь является возможность обращения данного процесса и перехода от рассмотрения произвольной последовательности из $n-1$ чисел (16) к рассмотрению ориентированного дерева, которое порождает эту последовательность. Действительно, рассмотрим произвольную последовательность x_1, x_2, $\ldots$, x_{n-1} чисел от 1 до n. Пусть V_1 — это наименьшее число, которое не представлено в последовательности $x_1, \ldots, x_{n-1}$, а V_2 — это наименьшее число $\neq V_1$, которое не представлено в последовательности $x_2, \ldots, x_{n-1}$ и т. д. Получив, таким образом, перестановку $V_1 V_2 \ldots V_n$ целых чисел $\{1, 2, \ldots, n\}$, проведем дуги от вершины V_j к вершине x_j для $1 \le j < n$ и получим ориентированный граф без ориентированных циклов, который согласно результату упр. 2.3.4.2–7 является ориентированным деревом. Ясно, что последовательность $x_1, x_2, \ldots, x_{n-1}$ тождественна последовательности (16) для этого ориентированного дерева.

Так как данный процесс является обратимым, получим взаимно однозначное соответствие между кортежами из $(n-1)$ чисел последовательности $\{1, 2, \ldots, n\}$ и ориентированными деревьями с этими вершинами. Следовательно, существует n^{n-1}

различных ориентированных деревьев с n помеченными вершинами. Если выбрать в качестве корня какую-либо одну вершину, причем безразлично, какую именно, то будет существовать n^{n-2} различных ориентированных деревьев с корнем, выбранным $\{1, 2, \ldots, n\}$. Для (15) это дает в итоге $16 = 4^{4-2}$ деревьев. Теперь легко определить количество *свободных деревьев* с помеченными вершинами (см. упр. 22). Количество *упорядоченных деревьев* с помеченными вершинами также легко подсчитать, если известен ответ для аналогичной задачи без помеченных вершин (см. упр. 23). Итак, задачи перечисления для трех фундаментальных классов деревьев с помеченными и непомеченными вершинами решены.

Интересно было бы узнать, что дает обычный метод производящих функций для решения задачи перечисления помеченных ориентированных деревьев. Для этого, вероятно, проще всего было бы рассмотреть величину $r(n, q)$, т. е. количество помеченных ориентированных графов с n вершинами и без ориентированных циклов, причем из q помеченных вершин этих графов выходит по одной дуге. Следовательно, количество помеченных ориентированных деревьев с указанным корнем равно $r(n, n-1)$. На основе таких обозначений и за счет простого подсчета аргументов получим, что для любого фиксированного целого числа m

$$r(n,q) = \sum_k \binom{q}{k} r(m+k,k)\, r(n-m-k,\, q-k), \quad \text{если } 0 \le m \le n-q, \tag{18}$$

$$r(n,q) = \sum_k \binom{q}{k} r(n-1,\, q-k), \quad \text{если } q = n-1. \tag{19}$$

Первое из этих соотношений получается, если разбить непомеченные вершины на две группы, A и B, с m вершинами в A и $n-q-m$ вершинами в B. Затем q помеченных вершин разобьем на k вершин, с которых начинаются пути, ведущие в A, и $q-k$ вершин, с которых начинаются пути, ведущие в B. Соотношение (19) получается, если рассмотреть ориентированные деревья, в которых корень имеет степень k.

Вид этих соотношений указывает на то, что в данном случае можно было успешно использовать производящую функцию:

$$G_m(z) = r(m,0) + r(m+1,\,1)z + \frac{r(m+2,\,2)z^2}{2!} + \cdots = \sum_k \frac{r(k+m,\,k)z^k}{k!}.$$

С помощью соотношения (18) получим $G_{n-q}(z) = G_m(z)G_{n-q-m}(z)$ и, следовательно, с помощью метода индукции по m получим $G_m(z) = G_1(z)^m$. Затем из соотношения (19) получим

$$G_1(z) = \sum_{n\ge1} \frac{r(n,\,n-1)z^{n-1}}{(n-1)!} = \sum_{k\ge0}\sum_{n\ge1} \frac{r(n-1,\,n-1-k)z^{n-1}}{k!\,(n-1-k)!}$$

$$= \sum_{k\ge0} \frac{z^k}{k!} G_k(z) = \sum_{k\ge0} \frac{\bigl(zG_1(z)\bigr)^k}{k!} = e^{zG_1(z)}.$$

Иначе говоря, для $G_1(z) = w$ решение этой задачи заключается в поиске коэффициентов решения такого трансцендентного уравнения:

$$w = e^{zw}. \tag{20}$$

Это уравнение можно было бы решить с помощью формулы обращения Лагранжа следующим образом. Из $z = \zeta/f(\zeta)$ следует, что

$$\zeta = \sum_{n\geq 1} \frac{z^n}{n!}\, g_n^{(n-1)}(0), \tag{21}$$

где $g_n(\zeta) = f(\zeta)^n$, если f — аналитическая функция в окрестности нуля и $f(0) \neq 0$ (см. упр. 4.7–16). В данном случае, предполагая, что $\zeta = zw$, $f(\zeta) = e^{\zeta}$, получим

$$w = \sum_{n\geq 0} \frac{(n+1)^{n-1}}{n!}\, z^n, \tag{22}$$

что полностью соответствует приведенному выше решению.

Дж. Н. Рейни (G. N. Raney) показал, что этот метод можно применить и для решения уравнения более общего вида

$$w = y_1 e^{z_1 w} + y_2 e^{z_2 w} + \cdots + y_s e^{z_s w},$$

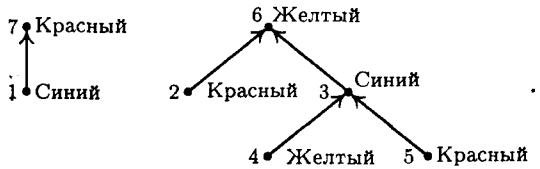
представив w в виде степенного ряда по $y_1, \ldots, y_s$ и $z_1, \ldots, z_s$. Для этого общего случая рассмотрим s-мерные векторы целых чисел

$$\mathbf{n} = (n_1, n_2, \ldots, n_s)$$

и запишем для удобства

$$\Sigma\mathbf{n} = n_1 + n_2 + \cdots + n_s.$$

Предположим, что имеется s цветов $C_1, C_2, \ldots, C_s$, и рассмотрим ориентированные графы, каждой вершине которых присвоен определенный цвет, например



(23)

Пусть $r(\mathbf{n}, \mathbf{q})$ — количество таких способов проведения дуг и присвоения цветов вершинам $\{1, 2, \ldots, n\}$, что

i) для $1 \leq i \leq s$ существует в точности n_i вершин с цветом C_i (следовательно, $n = \Sigma\mathbf{n}$);
ii) существует q дуг, которые выходят по одной из каждой вершины $\{1, 2, \ldots, q\}$;
iii) для $1 \leq i \leq s$ существует в точности q_i дуг, проходящих из вершин с цветом C_i (следовательно, $q = \Sigma\mathbf{q}$);
iv) не существует ориентированных циклов (следовательно, $q < n$ за исключением случая, когда $q = n = 0$).

Назовем это ($\mathbf{n}$,$\mathbf{q}$)-построением.

Например, если C_1 — красный цвет, C_2 — желтый цвет и C_3 — голубой цвет, то (23) схематично представляет $((3,2,2),(1,2,2))$-построение. При наличии только одного цвета получим уже решенную задачу для ориентированного дерева. Идея Рейни заключалась в обобщении одномерного построения до s размерностей.

Пусть $\mathbf{n}$ и $\mathbf{q}$ — фиксированные s-мерные векторы неотрицательных целых чисел, а $n = \sum\mathbf{n}$, $q = \sum\mathbf{q}$. Для каждого $(\mathbf{n},\mathbf{q})$-построения и каждого числа k, $1 \le k \le n$, определим *каноническое представление*, которое состоит из следующих четырех компонентов:

a) числа t, $q < t \le n$;
b) последовательности n цветов, в которой присутствует n_i цветов C_i;
c) последовательности q цветов, в которой присутствует q_i цветов C_i;
d) последовательности q_i элементов множества $\{1,2,\ldots,n_i\}$ для $1 \le i \le s$.

Это каноническое представление определяется следующим образом. Сначала перечислим вершины $\{1,2,\ldots,q\}$ в порядке $V_1, V_2, \ldots, V_q$ канонического представления ориентированных деревьев (как определено выше), а затем запишем под вершиной V_j номер вершины $f(V_j)$ на дуге, проходящей от V_j. Пусть $t = f(V_q)$, а последовательность цветов (c) соответствует цветам вершин $f(V_1),\ldots,f(V_q)$. Пусть последовательность цветов (b) соответствует цветам вершин $k, k+1, \ldots, n, 1, \ldots, k-1$. Наконец, пусть *i-я последовательность* (d) равна $x_{i1}, x_{i2}, \ldots, x_{iq_i}$, где $x_{ij} = m$, если j-й элемент цвета C_i последовательности $f(V_1),\ldots,f(V_q)$ является m-м элементом цвета C_i последовательности k, $k+1$, ..., n, 1, ..., $k-1$.

Например, рассмотрим графы (23) и допустим, что $k = 3$. Начнем с перечисления вершин $V_1,\ldots,V_5$ и $f(V_1),\ldots,f(V_5)$ и номеров под ними:

$$\begin{array}{ccccc} 1 & 2 & 4 & 5 & 3 \\ 7 & 6 & 3 & 3 & 6 \end{array}.$$

Значит, $t = 6$, а последовательность (c) представляет соответственно цвета вершин 7, 6, 3, 3, 6, а именно — красный, желтый, синий, синий, желтый. Последовательность (b) представляет соответственно цвета вершин 3, 4, 5, 6, 7, 1, 2, а именно — синий, желтый, красный, желтый, красный, синий, красный. Наконец, чтобы получить элементы такого цвета, нужно составить следующую таблицу.

Цвет	*Элементы такого цвета среди* 3,4,5,6,7,1,2	*Элементы такого цвета среди* 7,6,3,3,6	*Шифрование столбца 3 с помощью столбца 2*
Красный	5,7,2	7	2
Желтый	4,6	6,6	2,2
Синий	3,1	3,3	1,1

Итак, искомыми последовательностями типа (d) будут 2; 2,2 и 1,1.

На основе такого канонического представления можно восстановить исходное $(\mathbf{n},\mathbf{q})$-построение и число k, поступив следующим образом. Исходя из (a) и (c) узнаем цвет вершины t. Последний элемент последовательности (d) с таким цветом в сочетании с (b) позволяет узнать расположение числа t в последовательности $k,\ldots,n,1,\ldots,k-1$. Таким образом, узнаем значения k и цветов всех вершин. Тогда последовательности (d) вместе с (b) и (c) определят $f(V_1), f(V_2), \ldots, f(V_q)$

и наконец ориентированный граф будет полностью восстановлен после определения расположения $V_1, \ldots, V_q$ так, как это было сделано для ориентированных деревьев.

Обратимость подобного канонического представления позволяет подсчитать количество возможных (**n**,**q**)-построений, поскольку существует $n - q$ вариантов для (a),

$$\binom{n}{n_1, \ldots, n_s}$$

вариантов для (b),

$$\binom{q}{q_1, \ldots, q_s}$$

вариантов для (c) и $n_1^{q_1} n_2^{q_2} \ldots n_s^{q_s}$ вариантов для (d), которые выражены в виде полиномиальных коэффициентов. Общий результат найдем, разделив произведение полученных величин на n:

$$r(\mathbf{n}, \mathbf{q}) = \frac{n-q}{n} \frac{n!}{n_1! \ldots n_s!} \frac{q!}{q_1! \ldots q_s!} n_1^{q_1} n_2^{q_2} \ldots n_s^{q_s}. \tag{24}$$

Кроме того, можно получить аналогичные (18) и (19) соотношения:

$$r(\mathbf{n}, \mathbf{q}) = \sum_{\substack{\mathbf{k},\mathbf{t} \\ \sum(\mathbf{t}-\mathbf{k})=m}} \binom{\sum \mathbf{q}}{\sum \mathbf{k}} r(\mathbf{t}, \mathbf{k})\, r(\mathbf{n}-\mathbf{t}, \mathbf{q}-\mathbf{k}), \quad \text{если } 0 \le m \le \sum(\mathbf{n}-\mathbf{q}), \tag{25}$$

при условии, что $r(\mathbf{0}, \mathbf{0}) = 1$ и $r(\mathbf{n}, \mathbf{q}) = 0$, если значение n_i или q_i отрицательно либо если $q > n$;

$$r(\mathbf{n}, \mathbf{q}) = \sum_{i=1}^{s} \sum_{k} \binom{\sum \mathbf{q}}{k} r(\mathbf{n} - \mathbf{e}_i,\, \mathbf{q} - k\mathbf{e}_i), \quad \text{если } \sum \mathbf{n} = 1 + \sum \mathbf{q}, \tag{26}$$

где $\mathbf{e}_i$ — вектор с единицей в позиции i и нулями в остальных позициях. Соотношение (25) основано на разбиении вершин $\{q+1, \ldots, n\}$ на две части с m и $n-q-m$ элементами в каждой. Второе соотношение получается за счет удаления единственного корня и рассмотрения оставшейся после этого структуры. Рассмотрим теперь следующий результат.

Теорема R (George N. Raney, *Canadian J. Math.* **16** (1964), 755–762). *Пусть*

$$w = \sum_{\substack{\mathbf{n},\mathbf{q} \\ \sum(\mathbf{n}-\mathbf{q})=1}} \frac{r(\mathbf{n}, \mathbf{q})}{(\sum \mathbf{q})!} y_1^{n_1} \ldots y_s^{n_s} z_1^{q_1} \ldots z_s^{q_s}, \tag{27}$$

где $r(\mathbf{n}, \mathbf{q})$ *определяется формулой* (24), *а* $\mathbf{n}$, $\mathbf{q}$ — *s-мерные целочисленные векторы. Тогда w удовлетворяет тождеству*

$$w = y_1 e^{z_1 w} + y_2 e^{z_2 w} + \cdots + y_s e^{z_s w}. \tag{28}$$

Доказательство. С помощью (25) и метода индукции по m получим, что

$$w^m = \sum_{\substack{\mathbf{n},\mathbf{q} \\ \sum(\mathbf{n}-\mathbf{q})=m}} \frac{r(\mathbf{n}, \mathbf{q})}{(\sum \mathbf{q})!} y_1^{n_1} \ldots y_s^{n_s} z_1^{q_1} \ldots z_s^{q_s}. \tag{29}$$

Затем по формуле (26) находим, что

$$w = \sum_{i=1}^{s} \sum_{k} \sum_{\sum(\mathbf{n}-\mathbf{q})=1} \frac{r(\mathbf{n}-\mathbf{e}_i,\, \mathbf{q}-k\mathbf{e}_i)}{k!\,(\sum\mathbf{q}-k)!}\, y_1^{n_1}\dots y_s^{n_s}\, z_1^{q_1}\dots z_s^{q_s}$$

$$= \sum_{i=1}^{s} \sum_{k} \frac{1}{k!}\, y_i z_i^k \sum_{\sum(\mathbf{n}-\mathbf{q})=k} \frac{r(\mathbf{n},\mathbf{q})}{(\sum\mathbf{q})!}\, y_1^{n_1}\dots y_s^{n_s}\, z_1^{q_1}\dots z_s^{q_s}$$

$$= \sum_{i=1}^{s} \sum_{k} \frac{1}{k!}\, y_i z_i^k w^k. \quad \blacksquare$$

В приложениях особенно большое значение имеет специальный случай, когда $s = 1$ и $z_1 = 1$ в (27) и (28), а потому следующее выражение получило название "функция дерева" (tree function):

$$T(y) = \sum_{n\ge 1} \frac{n^{n-1}}{n!} y^n = y e^{T(y)}. \tag{30}$$

Некоторые замечательные свойства этой функции приведены в работе Корлеса, Гонне, Харе, Джефри, и Кнута [*Advances in Computational Math.* **5** (1996), 329–359].

Обзор формул перечисления деревьев, основанных на умелом применении производящих функций, предложен И. Дж. Гудом (I. J. Good) [*Proc. Cambridge Philos. Soc.* **61** (1965), 499–517; **64** (1968), 489]. Разработанная совсем недавно Андре Джойалем (André Joyal) математическая *теория видов* (*theory of species*) [*Advances in Math.* **42** (1981), 1–82] позволила расширить эту проблему до более высокого уровня, на котором алгебраические операции над производящими функциями непосредственно соответствуют комбинаторным свойствам структур. Многочисленные примеры из этой прекрасной и поучительной теории вместе с обобщениями многих полученных выше формул можно найти в книге *F. Bergeron, G. Labelle, and P. Leroux, Combinatorial Species and Tree-like Structures*, (Cambridge Univ. Press, 1998).

УПРАЖНЕНИЯ

1. [*M20*] (Задача Д. Пойа (G. Pólya).) Покажите, что

$$A(z) = z \cdot \exp\left(A(z) + \tfrac{1}{2}A(z^2) + \tfrac{1}{3}A(z^3) + \cdots\right).$$

[*Указание.* Возьмите логарифм выражения (3).]

2. [*HM24*] (Задача Р. Оттера (R. Otter).) Покажите, что числа a_n удовлетворяют такому соотношению:

$$na_{n+1} = a_1 s_{n1} + 2a_2 s_{n2} + \cdots + na_n s_{nn},$$

где

$$s_{nk} = \sum_{1\le j\le n/k} a_{n+1-jk}.$$

(Эти формулы очень полезны для вычисления a_n, так как $s_{nk} = s_{(n-k)k} + a_{n+1-k}$.)

3. [*M40*] Создайте программу определения количества непомеченных свободных деревьев и ориентированных деревьев с n вершинами для $n \le 100$. (Используйте результат упр. 2.) Исследуйте арифметические свойства этих чисел. Что можно сказать об их простых делителях или вычетах по модулю p?

▸ **4.** [*HM39*] (Задача Д. Пойа (G. Pólya), 1937.) С помощью теории функций комплексного переменного определите асимптотическое значение для количества ориентированных деревьев, выполнив приведенную ниже последовательность действий.

a) Покажите, что существует такое действительное число α от 0 до 1, для которого $A(z)$ имеет радиус сходимости α и $A(z)$ сходится абсолютно для всех таких комплексных z, что $|z| \le \alpha$, а максимальное значение $A(\alpha) = a < \infty$. [*Указание.* Если степенной ряд имеет неотрицательные коэффициенты, он либо является целой функцией, либо имеет особую точку на положительной полуоси действительных значений. Покажите, что $A(z)/z$ ограничено при $z \to \alpha - 0$, используя тождество из упр. 1.]

b) Пусть

$$F(z,w) = \exp\left(zw + \tfrac{1}{2}A(z^2) + \tfrac{1}{3}A(z^3) + \cdots\right) - w.$$

Покажите, что $F(z,w)$ в окрестности $(z,w) = (\alpha, a/\alpha)$ является аналитической функцией по каждой переменной в отдельности.

c) Покажите, что в точке $(z,w) = (\alpha, a/\alpha)$ имеем $\partial F/\partial w = 0$, а значит, $a = 1$.

d) В точке $(z,w) = (\alpha, 1/\alpha)$ покажите, что

$$\frac{\partial F}{\partial z} = \beta = \alpha^{-2} + \sum_{k\ge 2} \alpha^{k-2} A'(\alpha^k), \quad \text{а} \quad \frac{\partial^2 F}{\partial w^2} = \alpha.$$

e) Когда $|z| = \alpha$ и $z \ne \alpha$, покажите, что $\partial F/\partial w \ne 0$, а значит, $A(z)$ имеет только одну особую точку — $|z| = \alpha$.

f) Докажите, что существует область, бóльшая, чем $|z| < \alpha$, в которой

$$\frac{1}{z}A(z) = \frac{1}{\alpha} - \sqrt{2\beta(1 - z/\alpha)} + (1 - z/\alpha)R(z),$$

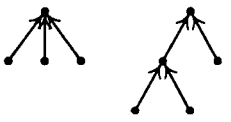
где $R(z)$ — аналитическая функция $\sqrt{z - \alpha}$.

g) Докажите, что в результате этого

$$a_n = \frac{1}{\alpha^{n-1} n}\sqrt{\beta/2\pi n} + O(n^{-5/2}\alpha^{-n}).$$

[*Замечание.* $1/\alpha \approx 2.955765285652$, $\alpha\sqrt{\beta/2\pi} \approx 0.439924012571$.]

▸ **5.** [*M25*] (Задача А. Кэли (A. Cayley).) Пусть c_n — количество непомеченных ориентированных деревьев с n листьями (т. е. с вершинами с нулевой степенью входа) и по крайней мере двумя поддеревьями во всех других вершинах. Значит, $c_3 = 2$, так как



Найдите формулу для производящей функции, аналогичную (3):

$$C(z) = \sum_n c_n z^n.$$

6. [*M25*] Пусть ориентированное бинарное дерево — это такое ориентированное дерево, в котором степень входа каждой вершины — не больше двух. Найдите сравнительно простое

соотношение, которое определяет производящую функцию $G(z)$ для количества различных ориентированных бинарных деревьев с n вершинами, и найдите несколько первых ее коэффициентов.

7. [*HM40*] Найдите асимптотическое значение для чисел из упр. 6 (см. упр. 4).

8. [*20*] Согласно (9) существует шесть свободных деревьев с шестью вершинами. Нарисуйте их и обозначьте вершины-центроиды.

9. [*M20*] Учитывая тот факт, что центроид может находиться не более чем в одном поддереве свободного дерева, докажите, что в свободном дереве может находиться не более двух центроидов и, более того, что если в таком дереве есть два центроида, то они будут смежными.

▸ **10.** [*M22*] Докажите, что свободное дерево с n вершинами и двумя центроидами состоит из двух свободных деревьев с $n/2$ вершинами, которые соединены одним ребром. И наоборот, если два свободных дерева с m вершинами соединить ребром, то получится свободное дерево с $2m$ вершинами и двумя центроидами.

▸ **11.** [*M28*] В данном разделе предложена формула (14) для того, чтобы найти количество различных бинарных деревьев с n узлами. Обобщите используемый в разделе метод вывода этой формулы для количества различных t-арных деревьев с n узлами. (См. упр. 2.3.1–35; *t-арное дерево либо пусто, либо состоит из корня и t непересекающихся t-арных деревьев.*) *Указание.* Используйте уравнение (21) из раздела 1.2.9.

12. [*M20*] Найдите количество помеченных ориентированных деревьев с n вершинами, используя детерминанты и результат упр. 2.3.4.2–19. (См. также упр.1.2.3–36.)

13. [*15*] Какое ориентированное дерево с вершинами $\{1, 2, \ldots, 10\}$ имеет такое каноническое представление: 3, 1, 4, 1, 5, 9, 2, 6, 5?

14. [*10*] Справедливо ли такое утверждение: "Последний элемент, $f(V_{n-1})$, канонического представления ориентированного дерева всегда является корнем этого дерева"?

15. [*21*] Изучите взаимосвязь (если таковая вообще существует) между алгоритмом топологической сортировки из раздела 2.2.3 и каноническим представлением ориентированного дерева.

16. [*25*] Создайте максимально эффективный алгоритм, который преобразует каноническое представление ориентированного дерева в обычное представление, используемое в компьютере с помощью связей PARENT.

▸ **17.** [*M26*] Пусть $f(x)$ — целозначная функция, где $1 \le f(x) \le m$ для всех целых $1 \le x \le m$. Определим, что $x \equiv y$, если $f^{[r]}(x) = f^{[s]}(y)$ для некоторого $r, s \ge 0$, где $f^{[0]}(x) = x$ и $f^{[r+1]}(x) = f(f^{[r]}(x))$. Используя методы перечисления, подобные приведенным в этом разделе, покажите, что количество таких функций, что $x \equiv y$ для всех x и y, равно $m^{m-1}Q(m)$, где $Q(m)$ является функцией, которая определена в разделе 1.2.11.3.

18. [*24*] Покажите, что следующий метод является еще одним способом определения взаимно однозначного соответствия между $(n-1)$-кортежами чисел от 1 до n и ориентированными деревьями с n помеченными вершинами. Пусть $V_1, \ldots, V_k$ — листья дерева в порядке возрастания. Пусть $(V_1, V_{k+1}, V_{k+2}, \ldots, V_q)$ — путь от V_1 к корню. В таком случае запишем вершины $V_q, \ldots, V_{k+2}, V_{k+1}$. Пусть $(V_2, V_{q+1}, V_{q+2}, \ldots, V_r)$ — такой кратчайший ориентированный путь от V_2, что V_r уже выписана. Затем выпишем $V_r, \ldots, V_{q+2}, V_{q+1}$. Далее пусть $(V_3, V_{r+1}, \ldots, V_s)$ — такой кратчайший ориентированный путь от V_3, что V_s уже выписана. После этого выпишем $V_s, \ldots, V_{r+1}$ и т. д. Например, дерево (17) могло бы быть выписано в таком виде: 3, 1, 3, 3, 5, 10, 5, 10, 1. Покажите, что этот процесс является обратимым, в частности нарисуйте схему ориентированного дерева с вершинами $\{1, 2, \ldots, 10\}$ и представлением 3, 1, 4, 1, 5, 9, 2, 6, 5.

19. [*M24*] Сколько существует различных помеченных ориентированных деревьев с n вершинами, среди которых k являются листьями (т. е. имеют нулевую степень входа)?

20. [*M24*] (Задача Дж. Риордана (J. Riordan).) Сколько существует различных помеченных ориентированных деревьев с n вершинами, из которых k_0 вершин имеют нулевую степень входа, k_1 — степень входа 1, k_2 — степень входа 2, ...? (Обратите внимание, что обязательно выполняются условия $k_0 + k_1 + k_2 + \cdots = n$ и $k_1 + 2k_2 + 3k_3 + \cdots = n - 1$.)

▸ **21.** [*M21*] Перечислите все помеченные ориентированные деревья, вершины которых имеют степень входа 0 или 2. (См. упр. 20 и 2.3-20.)

22. [*M20*] Сколько существует *помеченных* свободных деревьев с n вершинами? (Иначе говоря, используя n вершин, можно построить $2^{\binom{n}{2}}$ графов в зависимости от того, какие из $\binom{n}{2}$ возможных ребер используются в графе. Сколько среди них таких графов, которые являются свободными деревьями?)

23. [*M21*] Сколько упорядоченных деревьев можно построить с помощью n помеченных вершин? (Предложите простую формулу на основе факториалов.)

24. [*M16*] Все помеченные ориентированные деревья с вершинами 1–4 и корнем 1 показаны в виде схемы (15). Сколько можно получить помеченных *упорядоченных* деревьев с этими вершинами и этим корнем?

25. [*M20*] Чему равно значение $r(n, q)$ из уравнений (18) и (19)? (Предложите явную формулу, так как в данном разделе упоминается только соотношение $r(n, n-1) = n^{n-1}$.)

26. [*20*] На основе определения, приведенного в конце этого раздела, создайте $((3, 2, 4), (1, 4, 2))$-схему, аналогичную схеме (23), и найдите число k, которое соответствует каноническому представлению с $t = 8$, последовательностям цветов "красный, желтый, синий, красный, желтый, синий, красный, синий, синий" и "красный, желтый, синий, желтый, желтый, синий, желтый" и последовательностям чисел 3; 1, 2, 2, 1; 2, 4.

▸ **27.** [*M28*] Пусть $U_1, U_2, \ldots, U_p, \ldots, U_q$; $V_1, V_2, \ldots, V_r$ — вершины ориентированного графа, где $1 \le p \le q$. Пусть f — некоторая функция на множестве $\{p+1, \ldots, q\}$ с областью значений $\{1, 2, \ldots, r\}$ и пусть ориентированный граф содержит в точности $q - p$ дуг от U_k к $V_{f(k)}$ для $p < k \le q$. Покажите, что количество способов добавления r дополнительных дуг по одной от каждой вершины V к каждой вершине U, таких, что полученный в результате ориентированный граф не содержит ориентированных циклов, равно $q^{r-1}p$. Обобщая метод канонического представления, докажите это, т. е. установите взаимно однозначное соответствие между всеми такими способами добавления r дополнительных дуг и множеством всех последовательностей целых чисел $a_1, a_2, \ldots, a_r$, где $1 \le a_k \le q$ для $1 \le k < r$ и $1 \le a_r \le p$.

28. [*M22*] (*Двусторонние деревья.*) Используйте результат упр. 27 для перечисления таких помеченных свободных деревьев с вершинами $U_1, \ldots, U_m$, $V_1, \ldots, V_n$, в которых все ребра проведены от U_j к V_k для определенных j и k.

29. [*HM26*] Докажите, что если $E_k(r, t) = r(r + kt)^{k-1}/k!$ и $ze^z = \ln x$, то

$$x^r = \sum_{k \ge 0} E_k(r, t) z^k$$

для фиксированного t и для достаточно малых $|z|$ и $|x - 1|$. [Используйте тот факт, что $G_m(z) = G_1(z)^m$ в рассуждениях, приведенных после уравнения (19).] В этой формуле $\boldsymbol{r}$ обозначает произвольное действительное число. [*Замечание.* Как следствие этой формулы получим тождество

$$\sum_{k=0}^{n} E_k(r, t) E_{n-k}(s, t) = E_n(r + s, t),$$

из которого следует биномиальная теорема Абеля, см. (16) в разделе 1.2.6. Ср. также с (30) из того же раздела.]

30. [*M23*] Пусть $n, x, y, z_1, \ldots\ z_n$ — положительные целые числа. Рассмотрим множество $x + y + z_1 + \cdots + z_n + n$ вершин r_i, s_{jk}, t_j $(1 \le i \le x+y,\ 1 \le j \le n,\ 1 \le k \le z_j)$, для которых дуги проходят от s_{jk} к t_j для всех j и k. Согласно результату упр. 27 существует $(x+y)(x+y+z_1+\cdots+z_n)^{n-1}$ таких способов проведения дуг среди вершин $t_1, \ldots, t_n$, что полученный в результате ориентированный граф не будет содержать ориентированных циклов. Используйте этот результат для доказательства предложенного Гурвицем обобщения биномиальной теоремы.

$$\sum x(x+\epsilon_1 z_1+\cdots+\epsilon_n z_n)^{e_1+\cdots+e_n-1} y(y+(1-\epsilon_1)z_1+\cdots+(1-\epsilon_n)z_n)^{n-1-e_1-\cdots-e_n} = (x+y)(x+y+z_1+\cdots+z_n)^{n-1},$$

где сумма берется по всем 2^n вариантам наборов $\epsilon_1, \ldots, \epsilon_n$, состоящих из нулей и единиц.

31. [*M24*] Решите задачу из упр. 5 для упорядоченных деревьев, т. е. постройте производящую функцию для определения количества непомеченных упорядоченных деревьев с n концевыми узлами и без узлов со степенью 1.

32. [*M37*] (Задача А. Эрдейи (A. Erdélyi) и А. М. Х. Этерингтона (I. M. H. Etherington); см. *Edinburgh Math. Notes* **32** (1940), 7–12). Сколько существует упорядоченных и непомеченных деревьев с n_0 узлами со степенью 0, с n_1 узлами со степенью 1, ..., с n_m узлами со степенью m и без узлов со степенями выше m? (Решение этой задачи можно представить в явном виде с помощью факториалов и тем самым существенно обобщить результат упр. 11.)

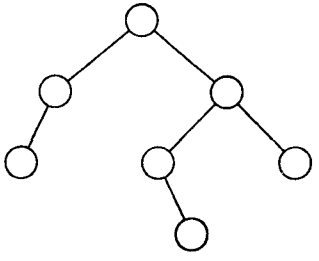
▸ **33.** [*M28*] В этом разделе в явном виде дано решение уравнения $w = y_1 e^{z_1 w} + \cdots + y_r e^{z_r w}$, которое основано на формулах перечисления для некоторых ориентированных лесов. Аналогично покажите, что формула перечисления из упр. 32 позволяет получить решение w уравнения

$$w = z_1 w^{e_1} + z_2 w^{e_2} + \cdots + z_r w^{e_r}$$

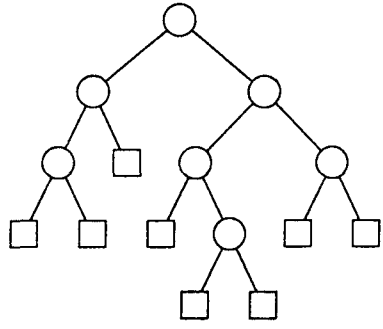
явно в виде степенного ряда $z_1, \ldots, z_r$. (Здесь $e_1, \ldots, e_r$ — фиксированные неотрицательные целые числа, среди которых по крайней мере одно равно нулю.)

2.3.4.5. Длина пути. Понятие "длина пути" дерева имеет большое значение для анализа алгоритмов, так как эта величина часто непосредственно связана со временем их выполнения. В таком смысле наибольший интерес вызывают бинарные деревья, поскольку они максимально близко отражают представление данных в компьютере.

Ниже в настоящем разделе мы будем рассматривать схемы бинарного дерева в расширенном виде: добавим к диаграмме дерева специальные узлы в местах, где в исходном дереве присутствуют пустые поддеревья, таким образом, что дерево



примет вид



(1)

Полученное дерево называется *расширенным бинарным деревом* (*extended binary tree*). После добавления квадратных узлов получим более удобную для работы структуру, которая будет довольно часто использоваться в последующих главах. Ясно, что каждый круглый узел имеет двух детей, а каждый квадратный — ни одного (ср. с 2.3–20). Если в дереве имеется n круглых и s квадратных узлов, то в нем имеется также $n + s - 1$ ребер (поскольку эта диаграмма — свободное дерево). Подсчитывая количество ребер другим способом, т. е. по количеству детей, получим $2n$ ребер. Отсюда следует, что

$$s = n + 1. \tag{2}$$

Иначе говоря, количество добавленных "внешних" узлов на единицу больше исходного количества "внутренних" узлов. (Другое доказательство приводится в упр. 2.3.1–14.) Формула (2) справедлива даже для $n = 0$.

Предположим, что бинарное дерево расширено именно таким образом. Тогда *длина внешнего пути дерева* (*external path length of the tree*), E, определяется, как сумма длин путей от корня к каждому узлу, взятая по всем внешним (квадратным) узлам. *Длина внутреннего пути дерева* (*internal path length*), I, определяется так же, но суммирование длин путей выполняется по внутренним (круглым) узлам. Для дерева (1) длина внешнего пути равна $E = 3+3+2+3+4+4+3+3 = 25$, а длина внутреннего пути равна $I = 2+1+0+2+3+1+2 = 11$. Эти две величины, очевидно, связаны формулой

$$E = I + 2n, \tag{3}$$

где n — количество внутренних узлов.

Для доказательства соотношения (3) удалим внутренний узел V, который находится на расстоянии k от корня, где оба ребенка вершины V — внешние узлы. Величина E при этом уменьшается на $2(k + 1)$, так как удаляются дети узла V, и в то же время увеличивается на k, так как узел V становится внешним. В целом, изменение величины E равно $-k - 2$, а изменение величины I равно $-k$. Далее справедливость формулы (3) доказывается по индукции.

Нетрудно видеть, что внутренняя длина пути (а значит, и внешняя длина пути) достигает наибольшего значения, когда дерево становится вырожденным, т. е. имеет линейную структуру. В этом случае длина внутреннего пути составляет

$$(n-1) + (n-2) + \cdots + 1 + 0 = \frac{n^2 - n}{2}.$$

Можно показать, что "средняя" длина пути для всех бинарных деревьев прямо пропорциональна $n\sqrt{n}$ (см. упр. 5).

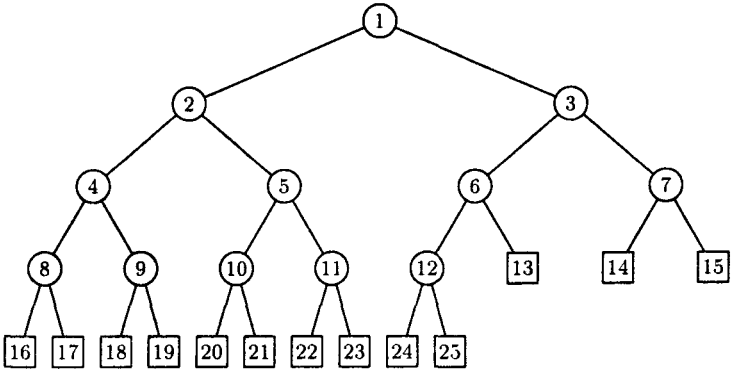
Рассмотрим теперь задачу построения бинарного дерева с n узлами, которое обладает *минимальной* длиной пути. Деревья такого типа имеют большое практическое значение, так как с их помощью можно сократить до минимума время выполнения различных алгоритмов. Ясно, что только один узел (корень) может находиться на нулевом расстоянии от корня. Далее, не более двух узлов может находиться на расстоянии 1 от корня, не более четырех узлов — на расстоянии 2, и т. д. Следовательно, *внутренняя длина пути всегда не меньше суммы первых n членов ряда*

$$0,\ 1,\ 1,\ 2,\ 2,\ 2,\ 2,\ 3,\ 3,\ 3,\ 3,\ 3,\ 3,\ 3,\ 3,\ 4,\ 4,\ 4,\ 4,\ldots .$$

Эту сумму можно выразить формулой $\sum_{k=1}^{n} \lfloor \lg k \rfloor$, которая, как нам известно из результата упр. 1.2.4–42, равна

$$(n+1)q - 2^{q+1} + 2, \qquad q = \lfloor \lg(n+1) \rfloor \tag{4}$$

Оптимальное значение (4) равно $n \lg n + O(n)$, так как $q = \lg n + O(1)$. Ясно, что этот результат достигается, например, в дереве следующего типа (здесь представлена схема дерева для $n = 12$).

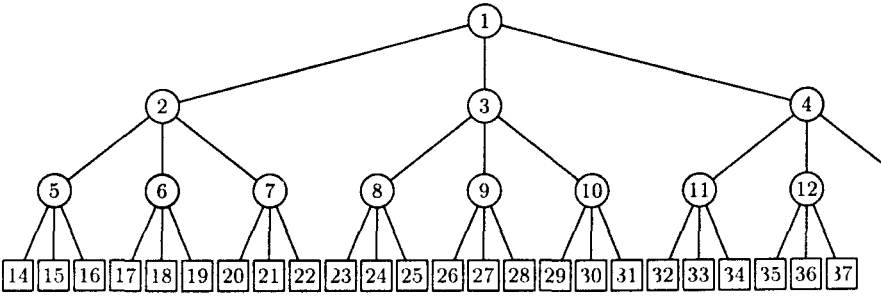


(5)

Дерево типа (5) называется *полным бинарным деревом* (*complete binary tree*) с n внутренними узлами. В общем случае можно перенумеровать внутренние узлы $1, 2, \ldots, n$. Эта нумерация удобна тем, что родителем узла k является узел $\lfloor k/2 \rfloor$, а детьми узла k — узлы $2k$ и $2k+1$. Внешние (концевые) узлы нумеруются соответственно числами от $n+1$ до $2n+1$ включительно

Отсюда следует, что полное бинарное дерево можно очень просто представить в последовательных ячейках памяти. причем его структура будет неявно задана не связями, а самим порядком расположения узлов. Полные бинарные деревья в явном или неявном виде применяются в очень многих важных компьютерных алгоритмах, поэтому читателю следует уделить им особое внимание

Рассматриваемые понятия можно обобщить для тернарных, кватернарных и других деревьев более высокого порядка. Определим *t-арное дерево* (*t-ary tree*) как множество узлов, которое либо пусто, либо состоит из узла и t упорядоченных и непересекающихся t-арных деревьев. (Это определение обобщает определение бинарного дерева из раздела 2.3.) *Полное тернарное дерево* (*complete ternary tree*) с 12 внутренними узлами имеет такой вид



(6)

Нетрудно видеть, что описанное выше построение обобщается для любого $t \geq 2$. В полном t-арном дереве с внутренними узлами $\{1, 2, \ldots, n\}$ родителем узла k будет узел

$$\lfloor (k + t - 2)/t \rfloor = \lceil (k - 1)/t \rceil,$$

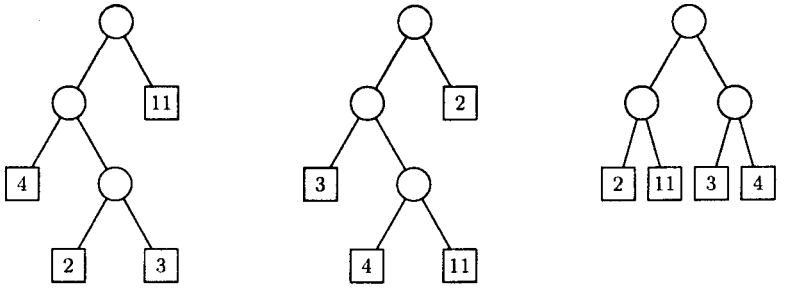
а детьми узла k — узлы

$$t(k-1)+2, \quad t(k-1)+3, \quad \ldots, \quad tk+1.$$

Это дерево имеет минимальную внутреннюю длину пути среди всех t-арных деревьев с n внутренними узлами. В упр. 8 приводится доказательство того, что его внутренняя длина пути равна

$$\left(n + \frac{1}{t-1}\right)q - \frac{(t^{q+1} - t)}{(t-1)^2}, \qquad q = \lfloor \log_t((t-1)n + 1) \rfloor. \tag{7}$$

Эти результаты имеют еще одно важное обобщение, если рассмотреть их с несколько другой точки зрения. Пусть даны m действительных чисел w_1, w_2, $\ldots$, w_m. Задача заключается в поиске расширенного бинарного дерева с m внешними узлами и такого соответствия между числами $w_1, \ldots, w_m$ и узлами этого дерева, при котором сумма $\sum w_j l_j$ является минимальной, где l_j — длина пути от корня, а сумма берется по всем внешним узлам. Например, если заданы числа 2, 3, 4, 11, можно построить три таких расширенных бинарных дерева.

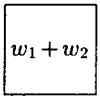


(8)

Здесь "взвешенными" длинами пути $\sum w_j l_j$ будут 34, 53 и 40 соответственно. (Как видно из данного примера, с помощью полностью сбалансированного дерева *нельзя* получить минимальное значение взвешенной длины пути для весов 2, 3, 4 и 11, хотя, как показано выше, это возможно в особом случае, с весами $w_1 = w_2 = \cdots = w_m = 1$.)

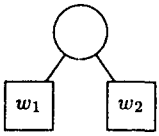
В разных компьютерных алгоритмах понятие взвешенной длины пути может интерпретироваться по-разному. Например, с его помощью можно выполнять слияние упорядоченных последовательностей с длинами $w_1, w_2, \ldots, w_m$ (см. главу 5). Одно из наиболее непосредственных приложений этого понятия заключается в том, что бинарное дерево рассматривается как некая программа поиска. Поиск начинается в корне с проверкой некоторого условия, затем в зависимости от его результата происходит переход к одной из двух ветвей, где снова проверяется некоторое условие, и т. д. Например, если необходимо выполнить проверку истинности четырех различных условий, а вероятности их истинности равны соответственно $\frac{2}{20}$, $\frac{3}{20}$, $\frac{4}{20}$ и $\frac{11}{20}$, то дерево, которое минимизирует взвешенную длину пути, и будет представлять собой *оптимальную программу поиска* (*optimal search procedure*). [Эти вероятности равны указанным в (8) весам, если умножить их на нормировочный множитель 20.]

Следующий элегантный алгоритм поиска дерева с минимальной взвешенной длиной пути был предложен Д. Хаффмэном [D. Huffman, *Proc. IRE* **40** (1952), 1098–1101]. Сначала нужно найти два наименьших веса w, например w_1 и w_2. После этого задача решается для $m-1$ весов w_1+w_2, w_3, $\ldots$, w_m, причем в ее решении узел



(9)

заменяется узлом

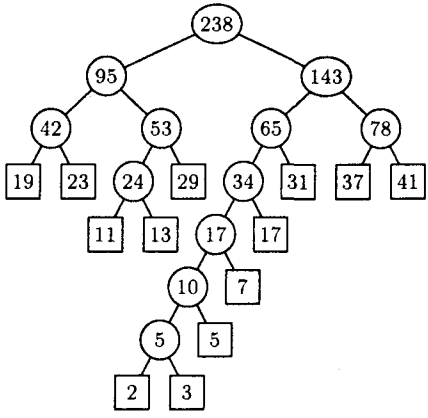


(10)

В качестве примера использования метода Хаффмэна найдем оптимальное дерево для весов 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41. Для этого сначала объединим вершины $2+3$ и найдем решение для $5, 5, 7, \ldots, 41$, затем объединим $5+5$ и т. д. В общем, последовательность действий будет выглядеть так.

2	3	5	7	11	13	17	19	23	29	31	37	41
	5	5	7	11	13	17	19	23	29	31	37	41
		10	7	11	13	17	19	23	29	31	37	41
			17	11	13	17	19	23	29	31	37	41
			17		24	17	19	23	29	31	37	41
					24	34	19	23	29	31	37	41
					24	34		42	29	31	37	41
						34		42	53	31	37	41
								42	53	65	37	41
								42	53	65		78
									95	65		78
									95			143
												238

Следовательно, такому построению Хаффмэна будет соответствовать дерево



(11)

(Числа в круглых узлах показывают связь между деревом и этапами приведенного выше вычисления; см. также упр. 9.)

Нетрудно доказать с помощью метода индукции по m, что этот способ действительно позволяет минимизировать взвешенную длину пути. Допустим, что даны веса $w_1 \le w_2 \le w_3 \le \cdots \le w_m$, где $m \ge 2$, и дерево, которое минимизирует взвешенную длину пути. (Такое дерево должно существовать, так как существует только конечное множество бинарных деревьев с m концевыми узлами.) Пусть V — внутренний узел, который находится на максимальном расстоянии от корня. Если веса w_1 и w_2 еще не приписаны детям узла V, то ими можно заменить величины, которые уже там находятся, не увеличивая взвешенную длину пути. Таким образом, существует дерево, которое минимизирует взвешенную длину пути и содержит поддерево (10). Теперь можно легко доказать, что взвешенная длина пути подобного дерева будет минимальной тогда и только тогда, когда это дерево с поддеревом (10), замененным узлом (9), обладает минимальной длиной пути для весов $w_1 + w_2$, $w_3, \ldots, w_m$. (см. упр. 9).

Всякий раз, когда в построении объединяются два веса, они по крайней мере не меньше весов, которые объединялись на предыдущем этапе, если все w_i — неотрицательные числа. Это значит, что существует прекрасный способ поиска дерева Хаффмэна при условии, что веса расположены в порядке неубывания. Тогда достаточно создать две очереди, одна из которых будет содержать исходные веса, а другая — объединенные веса. На каждом этапе этой процедуры наименьший неиспользованный вес будет находиться в начале одной из очередей, поэтому его не придется искать. В упр. 13 показано, как реализовать эту процедуру при работе с отрицательными весами.

Вообще, существует много деревьев, которые минимизируют $\sum w_j l_j$. Если в описанном выше алгоритме при очередном объединении весов всегда используется исходный, а не комбинированный вес, то полученное с помощью этого алгоритма дерево будет иметь наименьшее значение величин $\max l_j$ и $\sum l_j$ среди всех деревьев, которые минимизируют $\sum w_j l_j$. Если веса принимают положительные значения, это дерево также минимизирует $\sum w_j f(l_j)$ для *любой* выпуклой функции f по всем таким деревьям. [См. E. S. Schwartz, *Information and Control* **7** (1964), 37–44; G. Markowsky, *Acta Informatica* **16** (1981), 363–370.]

Метод Хаффмэна можно обобщить для t-арных, а также для бинарных деревьев (см. упр. 10). Еще одно важное обобщение метода Хаффмэна рассматривается в разделе 6.2.2. Обсуждение длины пути будет продолжено в разделах 5.3.1, 5.4.9 и 6.3.

УПРАЖНЕНИЯ

1. [*12*] Существуют ли какие-либо другие бинарные деревья с 12 внутренними узлами и минимальной длиной пути, кроме полного бинарного дерева (5)?

2. [*17*] Нарисуйте схему расширенного бинарного дерева с концевыми узлами, которые содержат веса 1, 4, 9, 16, 25, 36, 49, 64, 81, 100, имеющие минимальную взвешенную длину пути.

▸ **3.** [*M24*] Расширенное бинарное дерево с m внешними узлами определяет множество длин пути $l_1, l_2, \ldots, l_m$ от корня к соответствующим внешним узлам. И наоборот, если дано множество чисел $l_1, l_2, \ldots, l_m$, всегда ли можно построить расширенное бинарное дерево, в котором эти номера являются длинами пути, расположенными в некотором порядке? Покажите, что это возможно тогда и только тогда, когда $\sum_{j=1}^{m} 2^{-l_j} = 1$.

▸ **4.** [*M25*] (Задача Э. С. Шварца (E. S. Schwartz) и Б. Каллика (B. Kallick).) Предположим, $w_1 \le w_2 \le \cdots \le w_m$. Покажите, что существует расширенное бинарное дерево, которое минимизирует $\sum w_j l_j$ и для которого концевые узлы в порядке слева направо содержат значения $w_1, w_2, \ldots, w_m$. [Например, дерево (11) не удовлетворяет этому условию, так как веса в нем располагаются в порядке 19, 23, 11, 13, 29, 2, 3, 5, 7, 17, 31, 37, 41. Мы ищем дерево, в котором веса располагаются в порядке возрастания, а это условие не всегда выполняется в построении Хаффмэна.]

5. [*HM26*] Пусть

$$B(w,z) = \sum_{n,p\ge 0} b_{np} w^p z^n,$$

где b_{np} — количество бинарных деревьев с n узлами и внутренним путем длиной p. [Таким образом,

$$B(w,z) = 1 + z + 2wz^2 + (w^2 + 4w^3)z^3 + (4w^4 + 2w^5 + 8w^6)z^4 + \cdots;$$

$B(1,z)$ — функция $B(z)$ из уравнения (13) раздела 2.3.4.4.]

a) Найдите функциональное соотношение, которое характеризует $B(w,z)$, обобщив 2.3.4.4–(12).

b) Используйте результат (a) для определения *средней внутренней длины пути* бинарного дерева с n узлами, предполагая, что каждое из $\frac{1}{n+1}\binom{2n}{n}$ деревьев равновероятно.

c) Найдите асимптотическое значение этой величины.

6. [*16*] Какое соотношение, аналогичное соотношению (2), можно установить между количеством квадратных и круглых узлов, если t-арное дерево расширить квадратными узлами, как в (1)?

7. [*M21*] Какое соотношение можно установить между длинами внешнего и внутреннего путей в t-арном дереве? (См. упр. 6; необходимо получить обобщение уравнения (3).)

8. [*M23*] Докажите справедливость формулы (7).

9. [*M21*] Числа в круглых узлах дерева (11) равны сумме весов из внешних узлов соответствующих поддеревьев. Покажите, что сумма всех значений в круглых узлах равна взвешенной длине пути.

▸ **10.** [*M26*] (Задача Д. Хаффмэна.) Покажите, как можно построить t-арное дерево с минимальной взвешенной длиной пути для заданных неотрицательных весов $w_1, w_2, \ldots, w_m$. Постройте оптимальное тернарное дерево для весов 1, 4, 9, 16, 25, 36, 49, 64, 81, 100.

11. [*16*] Существует ли связь между полным бинарным деревом (5) и десятичной системой обозначений Дьюи для бинарных деревьев, описанной в упр. 2.3.1–5?

▸ **12.** [*M20*] Пусть случайным образом выбран некоторый узел бинарного дерева, причем такой выбор равновероятен для всех узлов дерева. Покажите, что средний размер поддерева с корнем в этом узле связан с длиной пути данного дерева.

13. [*22*] Создайте алгоритм, который на основе m весов $w_1 \le w_2 \le \cdots \le w_m$ приводит к построению расширенного бинарного дерева с минимальной взвешенной длиной пути. Представьте итоговое дерево в виде трех массивов

$$A[1]\ldots A[2m-1], \quad L[1]\ldots L[m-1], \quad R[1]\ldots R[m-1],$$

где $L[i]$ и $R[i]$ указывают на правых и левых детей внутреннего узла i, корнем является узел 1, а $A[i]$ — это вес узла i. Исходные веса следует представить в виде весов внешних узлов $A[m], \ldots, A[2m-1]$. При этом в созданном алгоритме должно выполняться менее чем $2m$ сравнений весов. *Замечание.* Некоторые или все из представленных весов могут иметь отрицательное значение!

14. [*25*] (Задача Т. Ч. Ху (T. C. Hu) и А. К. Такера (A. C. Tucker).) После k шагов алгоритма Хаффмэна объединенные узлы образуют лес из $m-k$ расширенных бинарных деревьев. Докажите, что этот лес имеет наименьшую общую взвешенную длину пути среди всех лесов, состоящих из $m-k$ расширенных бинарных деревьев с данными весами.

15. [*M25*] Покажите, что алгоритм, подобный алгоритму Хаффмэна, позволяет найти расширенное бинарное дерево, которое минимизирует величины

(a) $\max(w_1+l_1,\ldots,w_m+l_m)$

и

(b) $w_1x^{l_1}+\cdots+w_mx^{l_m}$ для заданного $x>1$.

16. [*M25*] (Задача Ф. К. Хвана (F. K. Hwang).) Пусть $w_1\le\cdots\le w_m$ и $w'_1\le\cdots\le w'_m$ — два множества весов с

$$\sum_{j=1}^{k}w_j\le\sum_{j=1}^{k}w'_j\qquad\text{для } 1\le k\le m.$$

Докажите, что минимальная взвешенная длина пути удовлетворяет неравенству

$$\sum_{j=1}^{m}w_jl_j\le\sum_{j=1}^{m}w'_jl'_j.$$

17. [*HM30*] (Задача Ч. Р. Глэсси (C. R. Glassey) и Р. М. Карпа (R. M. Karp).) Пусть $s_1,\ldots,s_{m-1}$ — числа во внутренних (круглых) узлах расширенного бинарного дерева, образованного с помощью алгоритма Хаффмэна, расположенные в порядке его построения. Пусть $s'_1,\ldots,s'_{m-1}$ — веса внутренних узлов произвольного расширенного бинарного дерева с тем же множеством весов $\{w_1,\ldots,w_m\}$, которые перечислены в таком произвольном порядке, что каждый некорневой внутренний узел располагается перед его родителем.

(a) Докажите, что

$$\sum_{j=1}^{k}s_j\le\sum_{j=1}^{k}s'_j\qquad\text{для } 1\le k<m.$$

(b) Результат (a) эквивалентен выражению

$$\sum_{j=1}^{m-1}f(s_j)\le\sum_{j=1}^{m-1}f(s'_j)$$

для каждой неубывающей вогнутой функции f, а именно — для каждой функции f с $f'(x)\ge0$ и $f''(x)\le0$. [См. Hardy, Littlewood, and Pólya, *Messenger of Math.* **58** (1939), 145–152.] Используйте этот факт, чтобы показать, что минимальное значение в рекуррентном соотношении

$$F(n)=f(n)+\min_{1\le k<n}\bigl(F(k)+F(n-k)\bigr),\qquad F(1)=0$$

всегда достигается для $k=2^{\lceil\lg(n/3)\rceil}$ при условии, что данная функция $f(n)$ обладает свойствами $\Delta f(n)=f(n+1)-f(n)\ge0$ и $\Delta^2f(n)=\Delta f(n+1)-\Delta f(n)\le0$.

***2.3.4.6. История и библиография.** Как известно, деревья появились уже на третий день сотворения мира и на протяжении веков древовидные структуры (особенно — *генеалогические* деревья) очень широко применялись. Как *математический* объект понятие дерева было впервые формально определено, по-видимому, в работе Г. Р. Кирхгофа (G. R. Kirchhoff) [*Annalen der Physik und Chemie* **72** (1847), 497–508, в английском переводе опубликовано в *IRE Transactions* **CT-5** (1958), 4–7]. Он

использовал свободные деревья для поиска набора фундаментальных циклов в электрической цепи с помощью законов, которые теперь носят его имя. Причем Кирхгоф делал это почти так же, как мы в разделе 2.3.4.1. Приблизительно в то же время понятие "дерево" появилось в книге К. Г. К. фон Штаудта (K. G. Chr. von Staudt) [*Geometrie der Lage* (pp. 20–21)]. Термин "дерево" и многие результаты, которые, в основном, имеют отношение к задаче перечисления деревьев, появились десять лет спустя в ряде работ Артура Кэли (Arthur Cayley) [см. *Collected Mathematical Papers of A. Cayley* **3** (1857), 242–246; **4** (1859), 112–115; **9** (1874), 202–204; **9** (1875), 427–460; **10** (1877), 598–600; **11** (1881), 365–367; **13** (1889), 26–28]. Кэли не знал о работах Кирхгофа и фон Штаудта и свои исследования начал, изучая структуру алгебраических формул. Позднее Кэли продолжил их в основном для изучения задач изомерии в химии. Древовидные структуры также независимо изучались К. В. Борхардтом (C. W. Borchardt) [*Crelle* **57** (1860), 111–121], И. Б. Листингом (J. B. Listing) [*Göttinger Abhandlungen*, Math. Classe, **10** (1862), 137–139] и М. Э. К. Жорданом (M. E. C. Jordan) [*Crelle* **70** (1869), 185–190].

"Лемма о бесконечном дереве" впервые была сформулирована Д. Кенигом (Dénes König) [*Fundamenta Math.* **8** (1926), 114–134]; особое место он уделил ей в главе 6 своей классической книги *Theorie der endlichen und unendlichen Graphen* (Leipzig, 1936). Аналогичный результат, так называемая "теорема о веере", чуть раньше был опубликован в работе Л. Э. Я. Брауэра (L. E. J. Brouwer) [*Verhandelingen Akad. Amsterdam* **12** (1919), 7], но автору пришлось использовать гораздо более "сильные" предположения. Работа Брауэра обсуждается в книге А. Хейтинга (A. Heyting) *Intuitionism* (1956), в разделе 3.4.

Формула (3) из раздела 2.3.4.4 для перечисления непомеченных ориентированных деревьев была предложена Кэли в его первой статье о деревьях. Во второй работе Кэли перечислил непомеченные упорядоченные деревья. Эквивалентная задача в геометрии (см. упр. 1) была поставлена и решена Й. А. фон Зегнером (J. von Segner) и Л. Эйлером (L. Euler) еще за сто лет до этого [*Novi Commentarii Academiæ Scientiarum Petropolitanæ* **7** (1758–1759), 13–15, 203–210]. К тому же ей были посвящены семь статей Г. Ламэ (G. Lamé), Э. Ш. Каталана (E. C. Catalan), Б. О. Родригеса (B. O. Rodrigues) и Ж. Ф. М. Бине (J. P. M. Binet) в *Journal de mathématiques* **3, 4** (1838, 1839). Дополнительные ссылки по этой теме можно найти в ответе к упр. 2.2.1–4. Соответствующие числа получили общепринятое название "числа Каталана" (Catalan numbers). Монголо-китайский математик Ан-Ту Минг (An-T'u Ming) рассматривал числа Каталана еще до 1750 года в исследованиях бесконечных рядов, но не связывал их с деревьями или другими комбинаторными объектами [см. J. Luo, *Acta Scientiarum Naturalium Universitatis Intramongolicæ* **19** (1988), 239–245; *Combinatorics and Graph Theory* (World Scientific Publishing, 1993), 68–70]. Числа Каталана возникают при изучении множества самых разнообразных задач. В великолепной книге Ричарда Стэнли (Richard Stanley) содержится около 60 примеров их применения [*Enumerative Combinatorics* **2** (Cambridge Univ. Press, 1999), Chapter 6]. Вероятно, наиболее удивительное из всех применений чисел Каталана связано с упорядочениями чисел, которые Г. С. М. Коксетер (H. S. M. Coxeter) из-за их симметрии назвал "узоры бордюра"(frieze patterns) (см. упр. 4).

Формула n^{n-2} для числа *помеченных* свободных деревьев была получена Сильвестром (J. J. Sylvester) [*Quart. J. Pure and Applied Math.* **1** (1857), 55–56] как побоч-

ный результат вычисления некоторого детерминанта (упр. 2.3.4.2–28). Независимо от него Кэли предложил еще один способ вывода данной формулы в 1889 году [см. приведенную выше ссылку]. В его весьма туманном обсуждении этого результата есть намек на связь между помеченными ориентированными деревьями и кортежами из $(n-1)$ чисел. Явное указание такой взаимосвязи было впервые опубликовано Хайнцем Прюфером (Heinz Prüfer) [*Arch. Math. und Phys.* **27** (1918), 142–144] совершенно независимо от работы Кэли. С тех пор этой теме было посвящено огромное количество публикаций. Прекрасный обзор классических результатов можно найти в книге Дж. В. Муна (J. W. Moon) *Counting Labelled Trees* (Montreal: Canadian Math. Congress, 1970).

Очень важная статья о перечислении деревьев и многих других видов комбинаторных структур опубликована Д. Пойа (G. Pólya) в *Acta Math.* **68** (1937), 145–253. Обсуждение задач перечисления для графов и прекрасная библиография ранних результатов по этой теме приводится в обзоре Фрэнка Харари (Frank Harary) [*Graph Theory and Theoretical Physics* (London: Academic Press, 1967), 1–41].

Метод поиска минимального значения взвешенной длины пути за счет повторного объединения наименьших весов был открыт Д. Хаффмэном (D. Huffman) [*Proc. IRE* **40** (1952), 1098–1101] в связи с задачей создания кодов для сжатия сообщений. Аналогичная идея была независимо предложена Сетом Циммерманом (Seth Zimmerman) [*AMM* **66** (1959), 690–693].

Несколько других достойных внимания статей о теории древовидных структур упомянуты в разделах 2.3.4.1–2.3.4.5 в связи с обсуждаемыми в них частными вопросами.

УПРАЖНЕНИЯ

▸ **1.** [*21*] Найдите простое взаимно однозначное соответствие между бинарными деревьями с n узлами и сечениями $(n+2)$-стороннего выпуклого многоугольника на n треугольников при условии, что стороны многоугольника различны.

▸ **2.** [*M26*] Т. П. Киркман (T. P. Kirkman) предположил в 1857 году, что количество способов проведения k неперекрывающихся диагоналей в r-стороннем многоугольнике равно $\binom{r+k}{k+1}\binom{r-3}{k}/(r+k)$.

a) Расширьте соотношение из упр. 1, чтобы можно было получить эквивалентную задачу о перечислении деревьев.

b) Докажите предположение Киркмана с помощью методов, описанных в упр. 2.3.4.4–32.

▸ **3.** [*M30*] Рассмотрите все способы такого разбиения вершин выпуклого n-угольника на k непустых частей, что ни одна из диагоналей между двумя вершинами одной части не пересекает диагональ между двумя вершинами другой части.

a) Найдите взаимно однозначное соответствие между непересекающимися разбиениями и каким-нибудь интересным классом древовидных структур.

b) Сколько существует способов выполнения такого разбиения для заданных n и k?

▸ **4.** [*M38*] (Задача Конвэя (Conway) и Коксетера (Coxeter).) *Узор бордюра* — это бесконечный массив наподобие

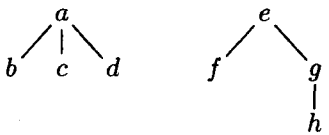
```
1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  ...
  3  1  3  1  4  1  2  3  1  3  1  4  1  2  3  1  3  1  4   ...
5  2  2  2  3  3  1  5  2  2  2  3  3  1  5  2  2  2  3   ...
  3  3  1  5  2  2  2  3  3  1  5  2  2  2  3  3  1  5  2   ... ,
1  4  1  2  3  1  3  1  4  1  2  3  1  3  1  4  1  2  3   ...
  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1   ...
```

в котором верхняя и нижняя строки полностью состоят из единиц, а каждый ромбик смежных значений $\begin{smallmatrix} & b & \\ a & & d \\ & c & \end{smallmatrix}$ удовлетворяет соотношению $ad - bc = 1$. Найдите взаимно однозначное соответствие между n-узловыми бинарными деревьями и $(n + 1)$-строчными узорами бордюра, которые состоят из положительных целых чисел.

2.3.5. Списки и "сборка мусора"

Почти в самом начале раздела 2.3 Список неформально определялся как "конечная последовательность атомов или Списков, число которых может быть больше нуля или равно нулю".

Любой лес является Списком, например лес

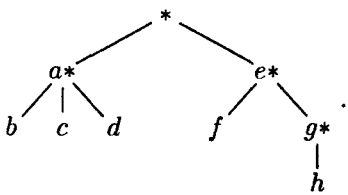


(1)

может рассматриваться как Список

$$\big(a\colon (b, c, d), e\colon (f, g\colon (h))\big), \tag{2}$$

а схема Списка будет выглядеть следующим образом:



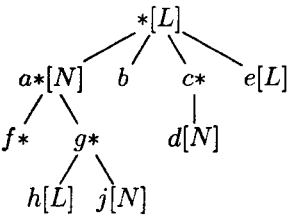
(3)

Читателю необходимо еще раз просмотреть предложенное ранее вводное описание Списков, в частности схемы (3)–(7), предложенные в самом начале раздела 2.3. Напомним, что в схеме (2) "$a\colon (b, c, d)$" означает, что (b, c, d) является Списком из трех атомов, который помечен атрибутом "a". Это обозначение совместимо с нашими общими представлениями о том, что в каждом узле дерева может, помимо структурных взаимосвязей, содержаться другая полезная информация. Однако, как и при обсуждении деревьев в разделе 2.3.3, вполне возможно, а иногда и очень желательно, использовать непомеченные Списки, чтобы вся информация содержалась в атомах.

Хотя любой лес может рассматриваться как Список, обратное утверждение неверно. Показанный ниже Список, вероятно, более типичен, чем (2) и (3), поскольку в нем показано, как могут нарушаться накладываемые на древовидную структуру ограничения:

$$L = \big(a\colon N, b, c\colon (d\colon N), e\colon L\big), \qquad N = \big(f\colon (\,), g\colon (h\colon L, j\colon N)\big). \tag{4}$$

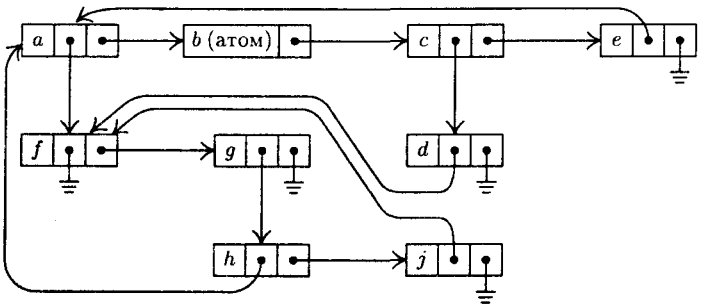
Схематически эту структуру можно представить в таком виде:



(5)

[Ср. со схемой 2.3–(7). При этом форму данных схем не стоит воспринимать слишком серьезно.]

Как и следовало ожидать, структуры наподобие Список могут быть отображены в памяти компьютера самыми разнообразными способами. Эти методы обычно являются вариациями на ту же основную тему, т. е. являются способами представления лесов деревьев общего типа на основе бинарных деревьев. Например, поле RLINK используется для указания следующего элемента Списка, а поле DLINK — для указания первого элемента подСписка. За счет естественного расширения представления в памяти, описанного в разделе 2.3.2, Список (5) можно отобразить так:



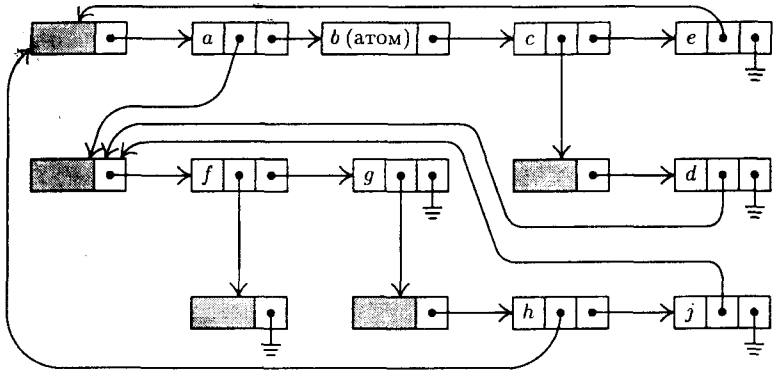
(6)

К сожалению, эта простая идея *не* совсем пригодна для наиболее распространенных приложений, в которых применяется обработка Списков. Предположим, например, что в Списке $L = (A, a, (A, A))$ содержатся три ссылки на Список $A = (b, c, d)$. Одна из типичных операций при обработке Списков заключается в удалении крайнего слева элемента Списка A таким образом, чтобы Список A после этого имел вид (c, d). Но для представления данного Списка в виде (6) в Списке L потребуется внести *три* изменения, поскольку каждый указатель на Список A указывает на удаляемый элемент b.

Немного поразмыслив, читатель, несомненно, придет к выводу, что было бы крайне нежелательно изменять указатели на каждую ссылку на Список A только потому, что удаляется первый элемент Списка A. (В этом примере можно попытаться схитрить и, предположив, что нет указателей на элемент c, сначала скопировать весь элемент c в ячейку, прежде занятую элементом b, а затем удалить старый элемент c. Однако эта уловка не поможет в ситуации, когда Список A утрачивает свой последний элемент и становится пустым.)

Поэтому вместо схемы представления (6) обычно используется другая схема, в которой в начале каждого Списка располагается *заголовок Списка* (*List head*) (о нем уже упоминалось в разделе 2.2.4). Каждый Список содержит дополни

тельный узел, который называется заголовком Списка, поэтому конфигурацию (6) можно представить, например, таким образом:



(7)

На самом деле использование узлов-заголовков не приводит к неэкономному расходованию памяти, так как в большинстве случаев другие, казалось бы, неиспользуемые поля (на схеме (7) они показаны в виде заштрихованных прямоугольников) этих узлов могут быть полезны в иных целях, например для счетчика ссылок, для указателя на правый конец Списка, для символьного имени, для "вспомогательного" поля, которое упрощает алгоритмы обхода деревьев, и т. д.

В исходной схеме (6) узел с элементом b является атомом, а узел с элементом f — пустым Списком. Эти два объекта структурно идентичны, поэтому читатель может вполне резонно возразить: "А зачем вообще нужно было упоминать об атомах?". Без утраты общности можно было бы определить Список всего лишь как "конечную последовательность атомов или Списков, число которых может быть больше нуля или равно нулю" с обычным соглашением о том, что в каждом узле Списка могут, помимо структурной информации, содержаться данные. Эта точка зрения вполне оправданна, а потому понятие "атом" представляется весьма искусственным. Однако для решения проблемы эффективного использования памяти компьютера было бы разумно выделить именно атомы, так как они не подвергаются тем универсальным операциям, которые желательно использовать для обработки Списков. Можно заметить, что в представлении наподобие (6) в каждом узле-атоме b содержится больше места для данных, чем в узле-Списке f. Кроме того, при использовании заголовков Списков в представлении наподобие (7) для узлов b и f предъявляются совершенно разные требования к памяти. Таким образом, концепция атомов вводится только для организации эффективного использования памяти компьютера. В типичных Списках содержится гораздо больше атомов, чем в приведенных здесь примерах. Примеры (4)–(7) даны лишь для того, чтобы продемонстрировать возможные усложнения данной схемы, а не присущую ей простоту.

Список, по сути, представляет собой линейный список, элементы которого могут содержать указатели на другие Списки. К наиболее распространенным и необходимым операциям со Списками обычно относятся операции с линейными списками (создание и удаление списка, вставка и удаление элементов, расщепление списка, сцепление списков), а также дополнительные операции, которые, прежде всего, интересны при работе с древовидными структурами (копирование, обход, ввод и вывод вложенной информации). Для этого можно использовать любой из трех основных

методов представления в памяти связанных линейных списков, а именно: обычный, циклический или дважды связанный список. В зависимости от используемого алгоритма эти способы представления могут иметь разные степени эффективности. Структура на схеме (7) с использованием этих типов списков в памяти компьютера может иметь следующее представление.

Адрес	*Обычный список*			*Циклический список*			*Дважды связанный список*			
	INFO	DLINK	RLINK	INFO	DLINK	RLINK	INFO	DLINK	LLINK	RLINK
010:	—	Заголовок	020	—	Заголовок	020	—	Заголовок	050	020
020:	*a*	060	030	*a*	060	030	*a*	060	010	030
030:	*b*	Атом	040	*b*	Атом	040	*b*	Атом	020	040
040:	*c*	090	050	*c*	090	050	*c*	090	030	050
050:	*e*	010	Λ	*e*	010	010	*e*	010	040	010
060:	—	Заголовок	070	—	Заголовок	070	—	Заголовок	080	070
070:	*f*	110	080	*f*	110	080	*f*	110	060	080
080:	*g*	120	Λ	*g*	120	060	*g*	120	070	060
090:	—	Заголовок	100	—	Заголовок	100	—	Заголовок	100	100
100:	*d*	060	Λ	*d*	060	090	*d*	060	090	090
110:	—	Заголовок	Λ	—	Заголовок	110	—	Заголовок	110	110
120:	—	Заголовок	130	—	Заголовок	130	—	Заголовок	140	130
130:	*h*	010	140	*h*	010	140	*h*	010	120	140
140:	*j*	060	Λ	*j*	060	120	*j*	060	130	120

(8)

Здесь LLINK используется для левого указателя в дважды связанном списке. Поля INFO и DLINK одинаковы для всех трех типов списков.

Нет необходимости повторять алгоритмы обработки Списков для любого из этих трех типов представления, так как они уже неоднократно обсуждались. Отметим лишь важные различия между Списками и более простыми типами списков, которые рассматривались выше.

1) В описанном выше типе представления в памяти узлы-атомы отличаются от других атомов. Более того, при использовании циклических и дважды связанных списков узлы-заголовки списков, которые применяются для упрощения обхода Списков, желательно отличать от других узлов. Поэтому каждый узел обычно имеет поле TYPE, в котором хранятся сведения о типе содержащихся в узле данных. Поле TYPE часто используется для того, чтобы можно было различать разные типы атомов (например, символьные величины, целочисленные величины либо числа с плавающей запятой при их обработке или отображении).

2) Формат узлов при выполнении общих операций со Списками в компьютере MIX может быть представлен следующими двумя способами.

а) Однословный формат, в котором все данные (INFO) содержатся в следующих атомах:

(9)

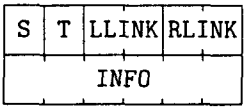
S: Знак (сокращение от слова “sign”), т. е. маркировочный бит, используемый при сборке мусора (см. ниже)

T: Тип (сокращение от слова “type”), т. е. T = 0 для заголовка Списка, T = 1 для элемента подСписка и T > 1 — для атомов

REF: Если T = 0, то REF — счетчик ссылок (см. ниже); если T = 1, то REF указывает на заголовок Списка рассматриваемого подСписка; если T > 1, то REF указывает на узел, в котором содержатся маркировочный бит и 5 байт атомарной информации

RLINK: Указатели, используемые в обычном или циклическом списке (8)

b) Двухсловный формат:



(10)

S, T: То же, что в (9)

LLINK, RLINK: Обычные указатели, которые используются в дважды связанных списках (8)

INFO: Полное слово с данными узла; в узле-заголовке оно может содержать счетчик ссылок; текущий указатель на внутренний компонент Списка, используемый для упрощения линейного обхода Списка; символьное имя и т. д. Если T = 1, то в нем содержится также поле DLINK

3) Ясно, что Списки представляют собой структуры очень общего типа. Действительно, по-видимому совершенно справедливо утверждение о том, что любую структуру, какой бы она ни была, можно представить в виде Списка, учитывая соответствующие соглашения. Благодаря такой универсальности Списков для упрощения работы с ними было разработано множество систем программирования. Причем для компьютера практически любого типа обычно существует сразу несколько таких систем. Они основаны на универсальном формате узлов, например на таком, как показано выше, на схемах (9) и (10), которые созданы специально для более гибкой организации операций со Списками. На самом деле ясно, что обычно такой универсальный формат — не самое лучшее решение какой-либо *конкретной* проблемы, поэтому универсальные программы выполняются значительно медленнее, чем система с настройками для конкретной задачи. Например, легко видеть, что если бы мы использовали универсальное представление Списка типа (9) или (10) практически во всех приложениях, с которыми нам приходилось работать до сих пор в этой главе, вместо принятого в них формата узлов, это существенно усложнило бы их код. Часто при обработке узлов Списка необходимо проверять содержимое поля T, что не требовалось в любой из перечисленных выше программ. Эта потеря эффективности при использовании универсальной системы во многих случаях компенсируется сравнительной простотой программирования и сокращением времени отладки.

4) Существует также чрезвычайно важное различие между алгоритмами обработки Списков и алгоритмами, приведенными выше в этой главе. Так как один Список может содержаться сразу в нескольких Списках, совершенно неясно, когда его следует вернуть в пул свободной памяти. До сих пор в наших алгоритмах всякий раз, когда узел NODE(X) уже не был нужен, упоминалась команда "AVAIL ⇐ X". Но поскольку во время работы Списки общего типа могут генерироваться и удаляться совершенно непредсказуемым образом, то часто очень трудно сказать, когда узел

уже не нужен. Следовательно, задача поддержания списка свободного пространства существенно усложняется при работе со Списками, чем при возникновении более простых случаев, которые рассматривались выше. Остальная часть этого раздела посвящается именно проблеме удаления неиспользуемых элементов и увеличения размера свободной памяти (storage reclamation problem).

Допустим, что нужно создать универсальную систему обработки Списков, которая будет применяться сотнями программистов. Для обслуживания списка свободной памяти существует два основных метода: *счетчиков ссылок* (*reference counters*) и *сборки мусора* (*garbage collection*). В методе на основе счетчиков ссылок в каждом поле используется новое поле с числом стрелок, которые указывают на этот узел. Показания такого счетчика легко отслеживать во время выполнения программы; при его обнулении узел можно поместить в список свободной памяти. С другой стороны, для метода сборки мусора требуется дополнительное однобитовое поле на каждом узле под названием *маркировочный бит* (*mark bit*). Основная идея в этом случае заключается в следующем: необходимо определить почти все алгоритмы так, чтобы они не возвращали неиспользуемые узлы в список свободной памяти сразу же. Пусть программа спокойно выполняется до тех пор, пока не будет израсходовано все свободное пространство в памяти. После этого специальный алгоритм "регенерации памяти" использует маркировочные биты для определения неиспользуемых в данный момент узлов и их возврата в список свободной памяти, после чего программа сможет продолжить свою работу.

Ни один из этих двух методов не является удовлетворительным. Основным недостатком метода счетчиков ссылок заключается в том, что не всегда удается освободить все неиспользуемые узлы. Он прекрасно подходит для перекрывающихся Списков (Списки могут содержать подСписки общего типа), но рекурсивные Списки наподобие L и N из (4) *никогда* не будут возвращены в список свободной памяти с помощью метода счетчика ссылок. Их счетчики будут всегда ненулевыми (поскольку они ссылаются на самих себя), даже если никакие другие Списки во время работы программы на них не указывают. Более того, для применения метода на основе счетчиков ссылок требуется использовать значительную часть памяти (хотя это пространство иногда все равно не используется из-за размера компьютерного слова).

Трудности, возникающие при использовании метода сборки мусора, связаны не только с нежелательной утратой одного бита в каждом узле, но и с тем, что он работает очень медленно при почти полном заполнении памяти. Причем в таких случаях количество найденных свободных ячеек памяти не оправдывает затраченных на это усилий. Программы, требующие больше свободной памяти (так происходит со многими не до конца отлаженными программами), часто очень неэкономно расходуют время при многократных вызовах программ сборки мусора, которые становятся почти бесполезными по мере исчерпания свободной памяти. Частичное решение этой проблемы заключается в том, что программист указывает некоторое значение k и работа прекращается, если в результате сборки мусора было найдено k или меньше неиспользуемых узлов.

Другая проблема заключается в трудности определения Списков, которые в текущий момент не являются "мусором". Если программист использует какие-то

нестандартные методы или хранит указатели в необычных местах, то очень велика вероятность того, что сборка мусора будет выполнена неверно. Некоторые самые загадочные происшествия при отладке программ были вызваны тем, что сборка мусора возникала неожиданно во время работы программ, которые раньше работали вполне благополучно. Для корректной сборки мусора часто требуется, чтобы программисты сохраняли во всех полях указателей только значащую информацию, хотя часто бывает удобно оставлять нетронутой бессмысленную информацию в полях, которые не используются программой (например, связь в последнем узле очереди; см. упр. 2.2.3–6).

Хотя для сборки мусора необходимо выделять по одному маркировочному биту для каждого узла, на самом деле можно было бы для этой же цели использовать отдельную таблицу собранных в другой области памяти маркировочных битов с заданным соответствием между расположением узла и его маркировочным битом. В некоторых компьютерах эта идея сводится к управлению мусорной кучей, что более привлекательно, чем управление отдельными битами в каждом узле.

Дж. Вейзенбаум (J. Weizenbaum) предложил интересную модификацию метода на основе счетчика ссылок. Используя дважды связанные Списки, он разместил счетчик только в заголовке каждого Списка. Таким образом, при перемещении по Списку указательные переменные не включаются в счетчики ссылок для отдельных узлов. Если известны правила, по которым поддерживаются счетчики ссылок для полных Списков, то теоретически известно, как избежать ссылок на любые Списки, счетчики ссылок которых равны нулю. Кроме того, можно явным образом сбросить счетчики циклов и возвратить отдельные Списки в область свободной памяти. При использовании этих идей следует соблюдать осторожность, поскольку в руках неопытных программистов они могут представлять опасность, а отладка программ заметно усложнится из-за наличия ссылок на уже удаленные узлы. Самой замечательной частью метода Вейзенбаума является способ работы со Списком, счетчик ссылок которого только что стал нулевым. Такой Список добавляется в *конце* текущего списка свободной памяти (например, для дважды связанного списка это можно очень просто сделать) и рассматривается как свободная память в последнюю очередь, только если использованы все предыдущие свободные ячейки памяти. В конечном счете, как только отдельные узлы Списка становятся свободными, значения счетчиков ссылок Списков, *на которые они* ссылаются, уменьшаются на один. Такое отложенное удаление Списков очень эффективно с точки зрения времени выполнения. Но при этом может возникнуть ситуация, когда некорректные программы какое-то время будут работать вполне корректно! Более подробное описание этого метода можно найти в *CACM* **6** (1963), 524–544.

Алгоритмы сборки мусора интересны сразу по нескольким причинам. Во-первых, они полезны в некоторых ситуациях, когда требуется пометить все узлы, прямо или косвенно ссылающиеся на данный узел. (Например, можно найти все подпрограммы, которые прямо или косвенно вызываются другой подпрограммой, как в упр. 2.2.3–26.)

Сборка мусора обычно состоит из двух фаз. Предположим, что маркировочные биты всех узлов в исходном состоянии равны нулю (или все они инициализируются нулями). Тогда во время первой фазы отметим все узлы, которые не относятся к мусору, начиная с тех, которые непосредственно доступны из основной программы.

На второй фазе выполняется последовательный проход всей области пула в памяти компьютера и все непомеченные узлы помещаются в список свободного пространства. Наиболее интересной является фаза маркировки, поэтому основное внимание будет уделено именно ей. Некоторые варианты второй фазы могут, однако, выглядеть весьма нетривиально (см. упр. 9).

При работе алгоритма сборки мусора для управления процессом маркировки доступна только очень ограниченная область памяти. Суть такой интригующей проблемы прояснится при дальнейшем изложении, но именно эту трудность многие не осознают при первом знакомстве с идеей сборки мусора. Причем в течение многих лет так и не было предложено достаточно хорошего решения этой проблемы.

Приведенный ниже алгоритм маркировки, вероятно, является наиболее очевидным.

Алгоритм А (*Маркировка*). Пусть вся используемая для Списка область памяти содержится в узлах NODE(1), NODE(2), ..., NODE(M). Предположим, что эти слова памяти являются либо атомами, либо полями связи ALINK и BLINK. Допустим, что все узлы в исходном состоянии *не маркированы.* Цель этого алгоритма заключается в *маркировке* всех узлов, к которым можно добраться с помощью цепочки указателей ALINK и/или BLINK в не являющихся атомами узлах, начиная с множества "непосредственно доступных" узлов, т. е. узлов, указатели на которые находятся в фиксированных ячейках памяти в основной программе. Эти фиксированные указатели используются в качестве отправной точки для доступа ко всем остальным данным.

A1. [Инициализация.] Пометить все "непосредственно доступные" узлы. Установить $K \leftarrow 1$.

A2. [Следует ли за узлом NODE(K) другой узел?] Установить $K1 \leftarrow K + 1$. Если NODE(K) — атом или немаркированный узел, перейти к шагу A3. В противном случае, если узел NODE(ALINK(K)) немаркированный, маркировать его и, если он не атом, установить $K1 \leftarrow \min(K1, ALINK(K))$. Аналогично, если NODE(BLINK(K)) не маркирован, маркировать его и, если он не атом, установить $K1 \leftarrow \min(K1, BLINK(K))$.

A3. [Готово?] Установить $K \leftarrow K1$. Если $K \leq M$, вернуться к шагу A2; в противном случае выполнение алгоритма прекращается. ▮

В этом и других алгоритмах настоящего раздела для удобства предполагается, что несуществующий узел NODE(Λ) является маркированным. (Например, ALINK(K) и BLINK(K) могут быть равны Λ на шаге A2.)

В одном из вариантов алгоритма А можно было бы установить $K1 \leftarrow M + 1$ на шаге A1, удалить операцию "$K1 \leftarrow K + 1$" на шаге A2, а шаг A3 использовать в следующей формулировке.

A3′. [Готово?] Установить $K \leftarrow K+1$. Если $K \leq M$, вернуться к шагу A2. В противном случае, если $K1 \leq M$, установить $K \leftarrow K1$ и $K1 \leftarrow M + 1$ и вернуться к шагу A2. Иначе — прекратить выполнение алгоритма.

Довольно трудно выполнить точный анализ алгоритма А или определить, лучше он или хуже только что описанного варианта, поскольку нет обоснованного способа описания распределения вероятностей самих исходных данных. Можно сказать, что

в худшем случае для выполнения этого алгоритма потребуется время, пропорциональное nM, где n — количество маркируемых ячеек. В общем, можно считать, что алгоритм работает очень медленно при больших значениях n. Для сборки мусора алгоритм A работает очень медленно, поэтому он не пригоден в качестве практичного метода решения этой задачи.

Другой очевидный алгоритм маркировки заключается в отслеживании всех путей и записи в стек сведений о точках разветвления этих путей.

Алгоритм B (*Маркировка*). Этот алгоритм позволяет получить те же результаты, что и алгоритм A, за счет использования ячеек `STACK[1]`, `STACK[2]`, ... в качестве вспомогательной памяти для хранения сведений обо всех путях, которые еще не были отслежены до конца.

B1. [Инициализация.] Пусть `T` — количество непосредственно доступных узлов. Маркировать их и разместить указатели на них в `STACK[1]`, ..., `STACK[T]`.

B2. [Стек пуст?] Если `T` = 0, выполнение алгоритма прекращается.

B3. [Удаление верхнего элемента стека.] Установить `K` ← `STACK[T]`, `T` ← `T` − 1.

B4. [Проверка связей.] Если `NODE(K)` — атом, вернуться к шагу B2. В противном случае, если `NODE(ALINK(K))` — непомеченный узел, его нужно маркировать и установить `T` ← `T` + 1, `STACK[T]` ← `ALINK(K)`; если `NODE(BLINK(K))` — немаркированный узел, то его нужно маркировать и установить `T` ← `T` + 1, `STACK[T]` ← `BLINK(K)`. Вернуться к шагу B2. ▮

Ясно, что время выполнения алгоритма B прямо пропорционально количеству маркируемых ячеек, причем это оценка наиболее благоприятного случая. На самом деле оказывается, что алгоритм не подходит для сборки мусора, потому что в нем не предусмотрено достаточно места для поддержания стека! Вполне разумно было бы предположить, что стек в алгоритме B может возрасти, например, до размера, равного 5% всего объема памяти. Но дело в том, что в процессе сборки мусора все свободное пространство уже израсходовано и остается только фиксированное (очень небольшое) количество ячеек, которые можно использовать в качестве такого стека. Бóльшая часть ранних программ сборки мусора основывалась на этом алгоритме, и при полном исчерпании специального используемого для стека пространства работа всей программы прекращалась.

Несколько лучший вариант основан на комбинации алгоритмов A и B с использованием стека фиксированного размера.

Алгоритм C (*Маркировка*). Этот алгоритм позволяет получить такой же результат, как и алгоритмы A и B, с помощью вспомогательной таблицы, состоящей из `H` ячеек, `STACK[0]`, `STACK[1]`, ..., `STACK[H − 1]`.

В данном алгоритме действие "Вставить `X` в стек" означает следующее: "Установить `T` ← (`T`+1) mod `H` и `STACK[T]` ← `X`. Если `T` = `B`, то установить `B` ← (`B`+1) mod `H` и `K1` ← min(`K1`, `STACK[B]`)". (Обратите внимание на то, что `T` указывает на текущий верхний элемент стека, а `B` — на одну позицию ниже текущего нижнего элемента стека. Таким образом, `STACK` функционирует, как дек с ограниченным входом.)

C1. [Инициализация.] Установить `T` ← `H` − 1, `B` ← `H` − 1, `K1` ← `M` + 1. Маркировать все непосредственно доступные узлы и последовательно внести их адреса в стек (как описано выше).

C2. [Стек пуст?] Если T = B перейти к шагу C5.

C3. [Удаление верхнего элемента.] Установить K ← STACK[T], T ← (T − 1) mod H.

C4. [Проверка связей.] Если NODE(K) — атом, вернуться к шагу C2. В противном случае, если NODE(ALINK(K)) не маркирован, маркировать его и вставить ALINK(K) в стек. Аналогично, если NODE(BLINK(K)) не маркирован, маркировать его и вставить BLINK(K) в стек. Вернуться к шагу C2.

C5. [Выметание мусора.] Если K1 > M, прекратить выполнение алгоритма. (Переменная K1 представляет наименьший адрес, по которому можно снова выйти на узел, подлежащий маркировке.) В противном случае, если NODE(K1) — атом или немаркированный узел, увеличить K1 на 1 и повторить этот шаг. Если NODE(K1) маркирован, то установить K ← K1, увеличить K1 на 1 и перейти к шагу C4. ▮

Этот алгоритм и алгоритм B можно усовершенствовать, если не вносить X в стек, когда NODE(X) — атом. Более того, на шагах B4 и C4 не следует помещать в стек элементы, которые сразу же будут удалены. Такие модификации достаточно просто выполняются, но они не использовались здесь, чтобы избежать усложнения алгоритмов.

Алгоритм C эквивалентен алгоритму A, когда H = 1, а также алгоритму B, когда H = M. Чем больше значение H, тем выше его эффективность. К сожалению, алгоритм C не поддается точному анализу по той же причине, что и алгоритм A. Поэтому не ясно, при каком значении H этот метод может быть оценен как достаточно быстрый. Можно считать значение H = 50 довольно правдоподобным, но не очень надежным критерием применяемости алгоритма C для сборки мусора в большинстве приложений.

В алгоритмах B и C стек располагается в последовательных ячейках памяти, но, как было показано выше в этой главе, методы связанного распределения памяти прекрасно подходят для организации стеков, которые располагаются в памяти непоследовательно. Таким образом, подразумевается, что стек в алгоритме B может располагаться *в той же области памяти, в которой происходит сборка мусора.* Это довольно легко можно сделать, если предоставить программе сборки мусора немного больше пространства в памяти. Предположим, например, что все Списки представлены в виде (9), но поля REF в узлах-заголовках Списков используются как сборщики мусора, а не как счетчики ссылок. Тогда алгоритм B можно перестроить, чтобы стек был организован с помощью полей REF в узлах-заголовках.

Алгоритм D (*Маркировка*). Этот алгоритм приводит к тому же результату, что и алгоритмы A, B и C, но предполагается, что в узлах вместо полей ALINK и BLINK содержатся описанные выше поля S, T, REF и RLINK. Поле S используется как маркировочный бит таким образом, что S(P) = 1 означает, что узел NODE(P) маркирован.

D1. [Инициализация.] Установить TOP ← Λ. Затем для каждого указателя P на заголовок непосредственно доступного Списка (см. шаг A1 алгоритма A), если S(P) = 0, установить S(P) ← 1, REF(P) ← TOP, TOP ← P.

D2. [Стек пуст?] Если TOP = Λ, прекратить выполнение алгоритма.

D3. [Удаление верхнего элемента.] Установить P ← TOP, TOP ← REF(P).

D4. [Перемещение по Списку.] Установить P $\leftarrow$ RLINK(P); затем, если P $= \Lambda$ или T(P) $= 0$, перейти к шагу D2. В противном случае установить S(P) $\leftarrow 1$. Если T(P) > 1, установить S(REF(P)) $\leftarrow 1$ (маркируя таким образом узел-атом). В противном случае (T(P) $= 1$); установить Q $\leftarrow$ REF(P); если Q $\neq \Lambda$ и S(Q) $= 0$, установить S(Q) $\leftarrow 1$, REF(Q) $\leftarrow$ TOP, TOP $\leftarrow$ Q. Повторить шаг D4. ▮

Алгоритм D можно сравнить с очень похожим на него алгоритмом B, время выполнения которого также прямо пропорционально количеству маркированных узлов. Однако алгоритм D *не* рекомендуется использовать без дополнительных оговорок, поскольку его, казалось бы, незначительные ограничения часто оказываются очень строгими для универсальных систем обработки Списков. В данном алгоритме требуется, чтобы все Списки были правильно организованы, например, как в (7), когда бы не начинался процесс сборки мусора. Но алгоритмы обработки Списков *на какое-то малое время* всегда искажают структуру Списков, и в такие моменты сборка мусора согласно алгоритму D должна быть приостановлена. Более того, следует внимательно выполнять действия на шаге D1, особенно тогда, когда в программе содержатся указатели на середину Списка.

Эти рассуждения приводят нас к алгоритму E (рис. 38), который представляет собой элегантный метод маркировки, независимо открытый Л. П. Дойчем (L. P. Deutsch), Г. Шорром (Herbert Schorr) и В. М. Вэйтом (W. M. Waite) в 1965 году. Используемые в нем предположения немного отличаются от тех, которые приняты в алгоритмах A–D.

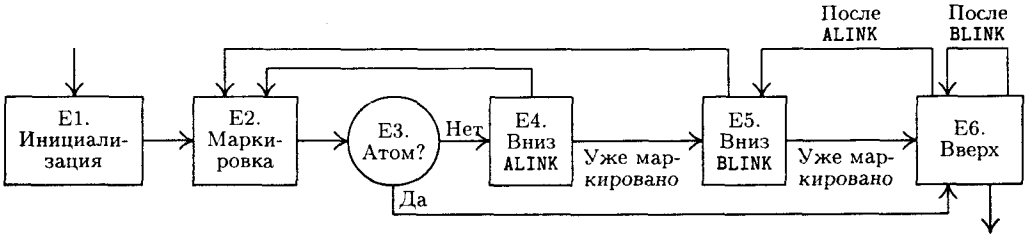


Рис. 38. Схема алгоритма E для маркировки без использования вспомогательного пространства стека.

Алгоритм E (*Маркировка*). Предположим, что дано множество узлов, содержащих следующие поля:

MARK (однобитовое поле),
ATOM (однобитовое поле),
ALINK (указательное поле),
BLINK (указательное поле).

Если ATOM $= 0$, то поля ALINK и BLINK могут содержать Λ или указатель на другой узел в таком же формате; если ATOM $= 1$, то содержание полей ALINK и BLINK не будет иметь никакого значения для этого алгоритма.

Для заданного ненулевого указателя P0 данный алгоритм устанавливает поле MARK равным 1 в узле NODE(P0) и во всех других узлах, до которых можно добраться от узла NODE(P0) по цепочке указателей ALINK и BLINK в узлах, где ATOM $=$ MARK $= 0$. В этом алгоритме используются три указательные переменные:

T, Q и P. Он модифицирует связи и контрольные биты таким образом, что после завершения работы алгоритма во всех полях ATOM, ALINK и BLINK восстанавливаются их исходные значения, хотя во время выполнения алгоритма они могут временно принимать другие значения.

E1. [Инициализация.] Установить T $\leftarrow \Lambda$, P $\leftarrow$ P0. (В остальной части этого алгоритма переменная T имеет двойственное значение. Если T $\neq \Lambda$, она указывает на верхний элемент того, что, по сути, является стеком в алгоритме D, иначе узел, на который указывает T, ранее содержал связь, равную P, вместо "искусственной" связи стека, и теперь находящуюся в NODE(T).)

E2. [Маркировка.] Установить MARK(P) $\leftarrow 1$.

E3. [Атом?] Если ATOM(P) $= 1$, то перейти к шагу E6.

E4. [Вниз по связям ALINK.] Установить Q $\leftarrow$ ALINK(P). Если Q $\neq \Lambda$ и MARK(Q) $= 0$, установить ATOM(P) $\leftarrow 1$, ALINK(P) $\leftarrow$ T, T $\leftarrow$ P, P $\leftarrow$ Q и перейти к шагу E2. (Здесь поле ATOM и поля ALINK временно изменяются таким образом, что структура Списка в некоторых маркированных узлах существенно меняется. Но на шаге E6 все будет восстановлено.)

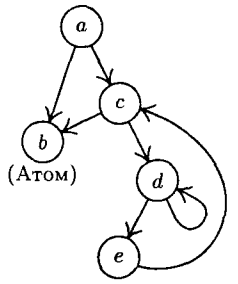
E5. [Вниз по связям BLINK.] Установить Q $\leftarrow$ BLINK(P). Если Q $\neq \Lambda$ и MARK(Q) $= 0$, то установить BLINK(P) $\leftarrow$ T, T $\leftarrow$ P, P $\leftarrow$ Q и перейти к шагу E2.

E6. [Вверх.] (На этом шаге отменяются переключения связей, выполненных на шаге E4 или E5; значение поля ATOM(T) указывает, какую из связей (ALINK(T) или BLINK(T)) следует восстановить.) Если T $= \Lambda$, выполнение алгоритма прекращается. В противном случае необходимо установить Q $\leftarrow$ T. Если ATOM(Q) $= 1$, то установить ATOM(Q) $\leftarrow 0$, T $\leftarrow$ ALINK(Q), ALINK(Q) $\leftarrow$ P, P $\leftarrow$ Q и вернуться к шагу E5. Если ATOM(Q) $= 0$, то установить T $\leftarrow$ BLINK(Q), BLINK(Q) $\leftarrow$ P, P $\leftarrow$ Q и повторить шаг E6. ▮

Пример использования этого алгоритма приведен на рис. 39, на котором показаны последовательные шаги работы со структурой простого Списка. Читателю будет полезно тщательно изучить алгоритм E. При этом следует обратить внимание на то, как искусственно меняется структура связей на шагах E4 и E5, чтобы организовать работу аналогично работе стека в алгоритме D. При возвращении в предыдущее состояние поле ATOM используется для указания того, какая из связей (ALINK или BLINK) содержит искусственную связь. "Вложения", показанные в нижней части рис. 39, демонстрируют, как каждый неатомный узел трижды посещается во время выполнения алгоритма E. При этом одна и та же конфигурация (T,P) встречается в начале шагов E2, E5 и E6.

Доказательство корректности алгоритма E можно сформулировать с помощью метода индукции по количеству узлов, которые необходимо маркировать. Попутно докажем, что в конце этого алгоритма переменной P возвращается ее исходное значение — P0 (см. упр. 3). Алгоритм E будет работать быстрее, если удалить шаг E3 и проверить условие "ATOM(Q) $= 1$" с выполнением соответствующих действий на шагах E4 и E5, а также проверить условие "ATOM(P0) $= 1$" на шаге E1. Этот алгоритм представлен именно в таком виде для упрощения изложения, а упомянутые выше модификации приведены в упр. 4.

Использованную в алгоритме E идею можно применить не только для решения проблем сборки мусора, но и для обхода деревьев, как в упр. 2.3.1-21. Читателю



a	ALINK[MARK]	*b*[0]	.	Λ[1]	.	*b*	.	.	.	.	.	.	.	.	.	.
	BLINK[ATOM]	*c*[0]	.	[1]	.	[0]	Λ	.	.	.	.	.	.	.	.	*c*
b	ALINK[MARK]	−[0]	.	.	[1]	.	.	.	.	.	.	.	.	.	.	.
	BLINK[ATOM]	−[1]	.	.	.	.	.	.	.	.	.	.	.	.	.	.
c	ALINK[MARK]	*b*[0]	.	.	.	.	[1]	.	.	.	.	.	.	.	.	.
	BLINK[ATOM]	*d*[0]	.	.	.	.	.	*a*	.	.	.	.	.	*d*	.	
d	ALINK[MARK]	*e*[0]	.	.	.	.	.	.	*c*[1]	.	.	*e*	.	.	.	
	BLINK[ATOM]	*d*[0]	.	.	.	.	.	.	[1]	.	.	[0]	.	.	.	
e	ALINK[MARK]	Λ[0]	.	.	.	.	.	.	.	[1]	.	.	.	.	.	
	BLINK[ATOM]	*c*[0]	.	.	.	.	.	.	.	.	.	.	.	.	.	

T	—	Λ	*a*	*a*	Λ	*a*	*a*	*c*	*d*	*d*	*d*	*c*	*c*	*a*	Λ
P	—	*a*	*b*	*b*	*a*	*c*	*c*	*d*	*e*	*e*	*e*	*d*	*d*	*c*	*a*
Следующий шаг	E1	E2	E2	E6	E5	E2	E5	E2	E2	E5	E6	E5	E6	E6	E6
Вложения															

Рис. 39. Структура, маркируемая алгоритмом E. (В таблице показаны только те изменения, которые произошли после предыдущего шага.)

будет полезно сравнить алгоритм E с более простой задачей, решение которой приводится в упр. 2.2.3–7.

Среди всех рассмотренных выше алгоритмов только алгоритм D можно непосредственно применять для Списков наподобие (9). В других алгоритмах выполняется проверка, не является ли данный узел P атомом, а условия (9) не совместимы с такой проверкой, потому что они допускают заполнение данными всего слова целиком, за исключением маркировочного бита. Однако другие алгоритмы можно модифицировать таким образом, что они смогут нормально функционировать, если данные атома можно будет отличить от указателя в связанном с ним слове, а не просматривать само слово целиком. В алгоритмах A и C можно довольно просто избежать маркировки слов-атомов, пока все слова-не атомы правильно маркированы. Тогда одного прохода по всем данным будет достаточно для маркировки всех слов-атомов. Алгоритм B можно еще проще модифицировать, так как для этого придется всего лишь сохранить слова-атомы вне стека. Почти так же просто можно адаптировать алгоритм E, хотя, если ALINK и BLINK указывают на данные атомы, потребуется в словах-не атомах ввести другое 1-битовое поле. Обычно это не так

трудно сделать. (Например, если узел состоит из двух слов, младший бит каждого поля связи можно использовать для хранения промежуточной информации.)

Хотя время выполнения алгоритма E прямо пропорционально количеству маркируемых узлов, константа пропорциональности не так мала, как для алгоритма B. Наиболее быстрый метод сборки мусора основан на сочетании алгоритмов B и E (подробности приводятся в упр. 5).

Попробуем теперь дать количественную оценку эффективности сборки мусора по сравнению с подходом на основе команды "AVAIL $\Leftarrow$ X", которая использовалась в большинстве предыдущих примеров настоящей главы. В каждом из этих случаев можно было бы опустить все особые упоминания о возврате узлов в свободную область памяти и использовать вместо них сборку мусора. (В специализированных приложениях по сравнению с множеством универсальных программ обработки Списков программирование и отладка программ сборки мусора выполняются гораздо сложнее, чем описанных до сих пор методов, и, конечно, для сборки мусора в каждом узле необходимо выделить дополнительный бит. Но в данном случае нас интересует сравнительная скорость выполнения уже готовых и отлаженных программ.)

Время выполнения наилучших программ сборки мусора можно представить в виде $c_1\mathtt{N} + c_2\mathtt{M}$, где c_1 и c_2 — константы, N — количество маркированных узлов, а M — общее количество узлов в памяти. Таким образом, M $-$ N — это количество найденных свободных узлов, а среднее время, необходимое для возврата одного узла в свободную область памяти, равно $(c_1\mathtt{N} + c_2\mathtt{M})/(\mathtt{M} - \mathtt{N})$. Пусть $\mathtt{N} = \rho\mathtt{M}$; тогда это выражение принимает вид $(c_1\rho + c_2)/(1 - \rho)$. Поэтому, если $\rho = \frac{3}{4}$, т. е. если память заполнена на три четверти, для возврата в свободную память одного узла потребуется $3c_1 + 4c_2$ единиц времени. А при заполнении $\rho = \frac{1}{4}$ соответствующие затраты времени составят всего $\frac{1}{3}c_1 + \frac{4}{3}c_2$ единиц времени. Если не использовать метод сборки мусора, то количество времени для возврата одного узла в свободную память будет равно некоторой константе c_3, а отношение c_3/c_1 вряд ли будет очень большим. Следовательно, нетрудно сделать вывод о том, насколько неэффективен метод сборки мусора при высокой степени заполнения памяти и соответственно насколько он эффективен при незначительном ее заполнении.

Для многих программ характерно то, что отношение количества маркированных узлов к общему объему памяти $\rho = \mathtt{N}/\mathtt{M}$ достаточно мало́. Если в таких случаях пул полностью заполняется, то лучше всего, наверное, переместить все активные Списки в другой пул такого же размера, используя упомянутый в упр. 10 метод копирования, но не беспокоясь о сохранении содержимого копируемых узлов. Тогда при заполнении второго пула можно снова переместить данные в первый пул. Благодаря такому методу в оперативной памяти можно одновременно хранить гораздо больше данных, поскольку поля связи указывали бы на соседние узлы. Более того, не потребовалось бы применять фазу маркировки, а распределение памяти было бы просто последовательным.

Сборку мусора можно использовать совместно с другими методами возврата ячеек в свободную память. Они не являются взаимно исключающими, и в некоторых системах применяются как метод счетчика ссылок так и метод сборки мусора, причем программист может удалять узлы даже явным образом. Основная идея в этом случае заключается в применении метода сборки мусора только "в крайнем

случае", когда все другие методы возврата неиспользуемых ячеек памяти уже не помогают. Хорошо продуманная система, в которой реализована эта идея и включен механизм отложенных операций со счетчиками ссылок для достижения повышенной эффективности, предложена Л. П. Дойчем (L. P. Deutsch) и Д. Г. Бобровым (D. G. Bobrow) [см. *CACM* **19** (1976), 522–526].

Кроме того, для Списков можно использовать последовательное представление, которое позволяет сэкономить пространство, занимаемое многими полями связи, за счет более сложного управления памятью. [См. N. E. Wiseman and J. O. Hiles, *Comp. J.* **10** (1968), 338–343; W. J. Hansen, *CACM* **12** (1969), 499–506; C. J. Cheney, *CACM* **13** (1970), 677–678.]

Даниэль П. Фридман (Daniel P. Friedman) и Дэвид С. Вайс (David S. Wise) обнаружили, что метод счетчика ссылок можно успешно применять даже в тех случаях, когда Списки указывают на самих себя, если только не учитывать при подсчете ссылок некоторые поля связи [*Inf. Proc. Letters* **8** (1979), 41–45].

В настоящее время накоплено огромное количество усовершенствованных вариантов алгоритмов сборки мусора. Подробный анализ всей научной литературы на эту тему, опубликованной до 1981 года, вместе с комментариями о дополнительных затратах на операции доступа к памяти при перекачках страниц памяти между оперативной и дисковой памятью можно найти в обзоре Ж. Коэна (Jacques Cohen) [*Computing Surveys* **13** (1981), 341–367].

Описанные здесь методы сборки мусора не совсем подходят для "интерактивных" приложений ("real time" applications), в которых все основные операции работы со Списками должны выполняться очень быстро. Даже если сборка мусора осуществляется не часто, все равно придется затратить довольно большую часть времени вычислений. В упр. 12 рассмотрены некоторые способы реализации метода сборки мусора в интерактивных приложениях.

Печально, что в наши дни
осталось так мало бесполезной информации.

— ОСКАР УАЙЛЬД (OSCAR WILDE) (1894)

УПРАЖНЕНИЯ

▸ **1.** [*M21*] В разделе 2.3.4 было показано, что "дерево" — это особый случай "классического" математического понятия "ориентированный граф". Можно ли описать Списки на основе терминологии теории графов?

2. [*20*] В разделе 2.3.1 показано, что обход дерева можно упростить, используя прошитое представление данных внутри компьютера. Можно ли аналогичным образом прошить структуру Списка?

3. [*M26*] Докажите корректность алгоритма E. [*Указание.* См. доказательство алгоритма 2.3.1T.]

4. [*28*] Создайте программу для компьютера MIX, которая реализует алгоритм E при условии, что узлы представлены одним словом компьютера MIX, причем маркировочный бит MARK находится в поле (0:0) ["+" = 0, "−" = 1], узел-атом ATOM — в поле (1:1), ALINK — в поле (2:3), BLINK — в поле (4:5), а $\Lambda = 0$. Определите также время выполнения этой программы на основе соответствующих параметров. (При работе с компьютером MIX не так уж просто определить, что содержится в ячейке памяти: −0 или +0. Поэтому данная особенность может существенным образом повлиять на вашу программу.)

5. [*25*] (Задача Шорра и Вэйта.) Предложите алгоритм маркировки, который представляет собой комбинацию алгоритмов B и E со следующими свойствами. Остаются в силе допущения алгоритма E в отношении полей внутри узлов и т. д. Но вспомогательный стек `STACK[1]`, `STACK[2]`, ..., `STACK[N]` используется так же, как в алгоритме B, и механизм алгоритма E применяется только при полном заполнении стека.

6. [*00*] При осуществлении количественной оценки в конце данного раздела сказано, что время выполнения программы сборки мусора приблизительно равно $c_1\mathtt{N} + c_2\mathtt{M}$ единицам. На каком основании в эту оценку включен член "$c_2\mathtt{M}$"?

7. [*24*] (Задача Р. В. Флойда (R. W. Floyd).) Создайте алгоритм маркировки, в котором так же, как и в алгоритме E, не используется вспомогательный стек, за исключением того, что (i) он имеет гораздо более сложное управление, поскольку в каждом узле содержатся поля `MARK`, `ALINK` и `BLINK`, а поля `ATOM`, которые помогают упростить управление, в нем отсутствуют; (ii) он имеет более простое управление, так как маркирует только бинарное дерево, а не Список общего типа. В этом случае связи `ALINK` и `BLINK` являются обычными связями `LLINK` и `RLINK` бинарного дерева.

▸ **8.** [*27*] (Задача Л. П. Дойча (L. P. Deutsch).) Создайте алгоритм маркировки, в котором так же, как в алгоритмах D и E, не используется вспомогательная память для стека. Однако измените этот метод так, чтобы его можно было применять для узлов переменного размера и для переменного количества указателей в следующем формате. В первом слове узла находятся два поля: `MARK` и `SIZE`; поле `MARK` имеет тот же смысл, что и в алгоритме E, а поле `SIZE` содержит число $n \geq 0$. Это значит, что вслед за первым словом последовательно располагается n слов, каждое из которых содержит поле `MARK` (которое равно нулю и должно оставаться таким) и поле `LINK` (которое равно Λ или указывает на первое слово другого узла). Например, узел с тремя указателями будет содержать четыре последовательно расположенных слова.

Первое слово	`MARK` = 0 (будет установлено равным 1)	`SIZE` = 3
Второе слово	`MARK` = 0	`LINK` = первый указатель
Третье слово	`MARK` = 0	`LINK` = второй указатель
Четвертое слово	`MARK` = 0	`LINK` = третий указатель

Созданный вами алгоритм должен маркировать все узлы, к которым можно добраться из заданного узла `P0`.

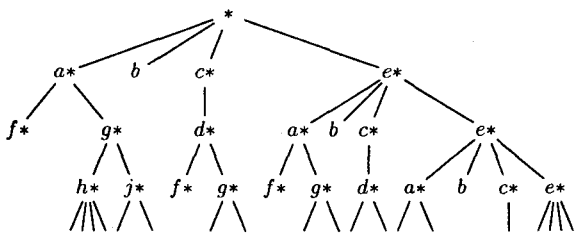
▸ **9.** [*28*] (Задача Д. Эдвардса (D. Edwards).) Создайте алгоритм для второй фазы сборки мусора, который "упаковывает память" в указанном ниже смысле. Пусть `NODE(1)`, ..., `NODE(M)` — однословные узлы с полями `MARK`, `ATOM`, `ALINK` и `BLINK`, как описано в алгоритме E. Предположим, что `MARK` = 1 во всех узлах, которые не относятся к мусору. Искомый алгоритм должен таким образом переместить маркированные узлы (в случае необходимости), чтобы они расположились в последовательных ячейках памяти `NODE(1)`, ..., `NODE(K)`. Причем в то же время поля `ALINK` и `BLINK` узлов, которые не являются атомами, необходимо изменить так (в случае необходимости), чтобы сохранилась структура Списка.

▸ **10.** [*28*] Создайте алгоритм для копирования структуры Списка, предполагая, что она имеет внутреннее представление наподобие (7). (Таким образом, при копировании с помощью этого алгоритма Списка, заголовок которого содержится в верхнем левом углу схемы (7), должно получиться новое множество Списков с 14 узлами со структурой и данными, которые идентичны показанным на схеме (7).)

Предположим, что структура Списка хранится в памяти и организована на основе полей `S`, `T`, `REF`, `RLINK` так, как на схеме (9), и что `NODE(P0)` — это заголовок копируемого Списка. Также допустим, что поле `REF` в заголовке каждого Списка равно Λ. Чтобы избежать дополнительных расходов памяти, в созданной вами программе копирования

должны использоваться поля REF (после завершения работы программы следует вернуть их исходные значения Λ).

11. [*M30*] Любая структура Списка может быть "полностью расширена" до древовидной структуры за счет повторения всех перекрывающихся элементов до тех пор, пока не останется ни одного такого элемента. Например, при расширении рекурсивного Списка таким образом можно получить бесконечное дерево: при расширении Списка (5) получится бесконечное дерево со следующими первыми четырьмя уровнями:



Создайте алгоритм для проверки *эквивалентности* двух структур Списка в том смысле, что древовидные структуры их полного расширения имеют одну и ту же форму. Например, Списки A и B в этом смысле эквивалентны, если

$$A = (a\colon C, b, a\colon (b\colon D));$$
$$B = (a\colon (b\colon D), b, a\colon E);$$
$$C = (b\colon (a\colon C));$$
$$D = (a\colon (b\colon D));$$
$$E = (b\colon (a\colon C)).$$

12. [*30*] (Задача М. Л. Мински (M. L. Minsky).) Покажите, что метод сборки мусора может вполне надежно использоваться в "оперативных" приложениях, например, когда компьютер управляет работой некоторого физического устройства, даже если на максимальное время выполнения каждой операции со Списками накладываются очень строгие ограничения. [*Указание.* Соблюдая все необходимые в таких случаях предосторожности, можно организовать сборку мусора и операции со Списками в параллельном режиме.]

2.4. МНОГОСВЯЗНЫЕ СТРУКТУРЫ

Теперь, после подробного исследования линейных списков и древовидных структур, принципы представления структурной информации внутри компьютера должны быть очевидны. В настоящем разделе будет рассмотрено еще одно приложение таких методов, на этот раз — для типичного случая с несколько более сложной структурной информацией. В приложениях более высокого уровня обычно используется сразу несколько типов структур.

"Многосвязная структура" состоит из узлов с несколькими полями связи в каждом узле, а не только с одним или двумя в структурах из предыдущих примеров. Примеры использования множественных связей приводились выше, например, при моделировании работы лифта в разделе 2.2.5 и работе с полиномами по многим переменным в разделе 2.3.3.

Как мы увидим далее, присутствие множества различных типов связей в одном узле *не* обязательно сопровождается усложнением их создания или восприятия по сравнению с уже рассмотренными алгоритмами. Кроме того, мы ответим на еще один важный вопрос: *В каком объеме структурная информация должна быть представлена в памяти в явном виде?*

Рассматриваемая здесь задача возникает в связи с созданием программы-компилятора для трансляции программ на языке COBOL и других сходных с ним языков. При работе с языком COBOL программист может присваивать символьные имена переменным программы на нескольких уровнях. Например, программа может иметь дело с файлами данных о продажах и покупках со следующей структурой.

```
1 SALES                    1 PURCHASES
  2 DATE                     2 DATE
    3 MONTH                    3 DAY
    3 DAY                      3 MONTH
    3 YEAR                     3 YEAR
  2 TRANSACTION              2 TRANSACTION
    3 ITEM                     3 ITEM
    3 QUANTITY                 3 QUANTITY
    3 PRICE                    3 PRICE
    3 TAX                      3 TAX
    3 BUYER                    3 SHIPPER
      4 NAME                     4 NAME
      4 ADDRESS                  4 ADDRESS
```
(1)

На этой схеме некоторой конфигурации данных показано, что каждый элемент файла SALES (продажи) состоит из двух частей: DATE (дата) и TRANSACTION (транзакция). Причем DATE подразделяется на три части, а TRANSACTION — на пять частей. Аналогичные замечания относятся и к файлу PURCHASES (покупки). Относительный порядок имен указывает порядок, в котором эти величины предстают во внешних представлениях файла (например, на магнитной ленте или распечатанных формах). Обратите внимание на то, что DAY и MONTH в этих двух файлах представлены в разном порядке. Программист приводит и другую не показанную здесь информацию, которая сообщает о том, какое пространство в памяти занимает каждый элемент данных и в каком формате эти данные представлены. Подобные соображения несущественны для темы данного раздела, а потому не будут рассматриваться.

Программист при работе с языком COBOL описывает сначала формат файла и другие переменные программы, а затем — алгоритмы, которые оперируют этими величинами. Для ссылки на отдельную переменную в приведенном выше примере было бы недостаточно просто указать имя DAY, так как не существует способа указания, в каком файле она находится: в SALES или PURCHASES. Следовательно, при работе с языком COBOL можно с помощью выражения DAY OF SALES указать, что элемент DAY является частью элемента SALES. Программист мог бы также записать в более полной форме, что

DAY OF DATE OF SALES,

но, вообще-то, не следует придавать величинам больше квалификаций (описаний), чем это действительно необходимо, во избежание неоднозначности. Таким образом, выражение

NAME OF SHIPPER OF TRANSACTION OF PURCHASES

можно сократить до

NAME OF SHIPPER,

поскольку существует только одна часть данных с именем SHIPPER.

Эти правила языка COBOL могут быть выражены более точно в следующей форме.

a) Каждому имени предшествует некоторое связанное с ним положительное целое число, которое называется *номером уровня*. Имя относится либо к *простейшему элементу*, либо к *группе* из одного или нескольких элементов со своими именами. В последнем случае все элементы группы должны иметь один номер уровня, который должен быть выше, чем номер уровня для имени группы. (Например, элементы DATE и TRANSACTION в приведенном примере имеют уровень 2, который выше уровня 1 для элемента SALES.)

b) Для ссылки на простейший элемент или группу элементов с именем A_0 используется общая форма

$$A_0 \text{ OF } A_1 \text{ OF } \ldots \text{ OF } A_n,$$

где $n \geq 0$, а A_j является именем элемента, который прямо или косвенно содержится внутри группы с именем A_{j+1} для $0 \leq j < n$. Должен существовать только один элемент A_0, который удовлетворяет этому условию.

c) Если одно и то же имя A_0 появляется в нескольких местах, должен существовать способ ссылки на каждый случай использования такого имени с помощью квалификации (описания).

Например, согласно правилу (c) конфигурация данных

```
1 AA
  2 BB
    3 CC
    3 DD
  2 CC
```
(2)

недопустима, так как при втором появлении элемента CC не существует однозначного способа ссылки на элементы с таким именем (см. упр. 4).

COBOL обладает еще одним свойством, которое может оказать влияние на процесс создания компилятора и работу рассматриваемых приложений, а именно — возможностью ссылаться сразу на несколько элементов. В таком случае программист может записать

MOVE CORRESPONDING α TO β,

что приведет к перемещению всех элементов с соответствующими именами из области данных α в область данных β. Например, команда языка COBOL

```
MOVE CORRESPONDING DATE OF SALES TO DATE OF PURCHASES
```

означает, что значениями переменных MONTH, DAY и YEAR из файла SALES нужно заменить значения переменных DAY, MONTH, YEAR в файле PURCHASES. (Относительный порядок DAY и MONTH при этом изменяется.)

В данном разделе будут рассмотрены три алгоритма, которые можно использовать в компиляторе COBOL и которые предназначены для выполнения перечисленных ниже действий.

Операция 1. Обработать описания имен и номеров уровней, подобных показанным на схеме (1), разместив соответствующую информацию в таблицах внутри компилятора для использования в операциях 2 и 3.

Операция 2. Определить, справедлива ли заданная ссылка, квалифицированная согласно правилу (b), и, если справедлива, найти соответствующий элемент данных.

Операция 3. Найти все соответствующие пары элементов, которые указаны в команде CORRESPONDING.

Допустим, что наш компилятор уже имеет "подпрограмму таблиц символов", которая может преобразовать символьное имя в связь, указывающую на позицию таблицы, соответствующую этому имени. (Более подробно методы построения алгоритмов для обработки таблиц символов обсуждаются в главе 6.) Помимо таблицы символов, имеется таблица большего размера, *таблица данных*, содержащая по одной позиции для каждого элемента данных из исходной программы на языке COBOL, которую нужно откомпилировать.

Очевидно, что нельзя создать алгоритм выполнения операции 1 до тех пор, пока неизвестно, какого рода информацию предполагается хранить в таблице данных. Причем форма таблицы данных зависит от типа информации, необходимой для выполнения операций 2 и 3. Таким образом, прежде всего следует обратить внимание на операции 2 и 3.

Для определения значения ссылки на языке COBOL

$$A_0 \text{ OF } A_1 \text{ OF } \ldots \text{ OF } A_n, \qquad n \geq 0, \tag{3}$$

сначала необходимо найти имя A_0 в таблице символов. Причем должен существовать ряд связей от позиции таблицы символов ко всем позициям таблицы данных, которые относятся к этому имени. Затем для каждой позиции таблицы данных потребуется установить связь с элементом-группой, в которую он входит. Тогда, если существует поле связи от позиций таблицы данных к таблице символов, нетрудно сообразить, как следует организовать обработку ссылок наподобие (3). Более того, чтобы найти пары, заданные в команде MOVE CORRESPONDING, потребуется установить некоторые связи от позиций таблицы данных для каждого элемента-группы к отдельным элементам этой группы.

Таким образом, для каждой позиции таблицы данных необходимо создать дополнительно пять полей связи:

PREV (связь с предыдущей позицией с тем же именем, если таковая имеется);

PARENT (связь с наименьшей группой, если таковая имеется, содержащей элемент);

NAME (связь с позицией таблицы символов элемента);

CHILD (связь с первым подэлементом группы);

SIB (связь со следующим подэлементом группы, содержащей элемент).

Ясно, что структуры данных в языке COBOL, подобные приведенным выше структурам SALES и PURCHASES, являются деревьями, а связи наподобие PARENT, CHILD и SIB уже знакомы нам из предыдущего материала. (Представление дерева в виде обычного бинарного дерева основано на связях CHILD и SIB, а при добавлении связи PARENT получим "трижды связанное дерево". Пять упомянутых выше связей состоят из этих трех связей вместе со связями PREV и NAME, которые несут дополнительную информацию о данной древовидной структуре.)

Вероятно, не все пять связей являются необходимыми, или достаточными, но попробуем создать алгоритм с исходным предположением о том, что элементы таблицы данных содержат все пять полей (и дополнительную информацию, которая не имеет отношения к данной проблеме). В качестве примера множественного связывания рассмотрим такие две структуры данных языка COBOL:

```
1 A          1 H
  3 B          5 F
    7 C          8 G
    7 D        5 B
  3 E          5 C
  3 F            9 E
    4 G          9 D
                 9 G
```
(4)

Их следует представить в виде (5) (со связями, указанными в символьной форме). Поле LINK в каждой позиции таблицы символов указывает на последнюю из встреченных позиций таблицы данных с символьным именем из позиции таблицы символов.

Сначала потребуется создать алгоритм для построения таблицы данных такого типа. Обратите внимание на то, что в языке COBOL предусмотрена гибкость выбора номеров уровней. Левая структура (4) полностью эквивалентна структуре

```
1 A
  2 B
    3 C
    3 D  ,
  2 E
  2 F
    3 G
```

поскольку номера уровней необязательно должны быть последовательными числами.

Таблица символов

	LINK
A:	A1
B:	B5
C:	C5
D:	D9
E:	E9
F:	F5
G:	G9
H:	H1

В пустых клетках содержится информация, не имеющая отношения к данной задаче

Таблица данных

	PREV	PARENT	NAME	CHILD	SIB	
A1:	Λ	Λ	A	B3	H1	
B3:	Λ	A1	B	C7	E3	
C7:	Λ	B3	C	Λ	D7	
D7:	Λ	B3	D	Λ	Λ	
E3:	Λ	A1	E	Λ	F3	
F3:	Λ	A1	F	G4	Λ	
G4:	Λ	F3	G	Λ	Λ	
H1:	Λ	Λ	H	F5	Λ	
F5:	F3	H1	F	G8	B5	
G8:	G4	F5	G	Λ	Λ	
B5:	B3	H1	B	Λ	C5	
C5:	C7	H1	C	E9	Λ	
E9:	E3	C5	E	Λ	D9	
D9:	D7	C5	D	Λ	G9	
G9:	G8	C5	G	Λ	Λ	

(5)

Однако некоторые последовательности номеров уровней недопустимы. Например, если номер уровня для элемента D в (4) был бы заменен номером 6 (в любом месте), была бы получена бессмысленная конфигурация данных, нарушающая правило, в соответствии с которым все элементы группы должны иметь одинаковые номера. Поэтому в следующем алгоритме выполняется проверка, соблюдается ли правило (а) языка COBOL.

Алгоритм А (*Построение таблицы данных*). Этот алгоритм позволяет получить последовательность пар (L, P), где L — положительное целое число, обозначающее номер уровня, а P — позиция таблицы символов, соответствующая таким структурам данных COBOL, как (4). Данный алгоритм создает таблицу данных, подобную приведенной выше, в примере (5). Когда P указывает на позицию таблицы символов, которая прежде не встречалась, связь LINK(P) становится равной Λ. В этом алгоритме используется вспомогательный стек, который обрабатывается, как обычный стек (на основе последовательного распределения памяти, как в разделе 2.2.2, или на основе связанного распределения памяти, как в разделе 2.2.3).

A1. [Инициализация.] Ввести в стек элемент $(0, \Lambda)$. (В этом алгоритме стек будет содержать пары (L, P), где L — целое число, а P — указатель. В ходе работы алгоритма стек содержит номер уровня и указатели на последние позиции данных на всех уровнях данного дерева, которые располагаются выше текущего уровня. Например, в приведенном примере до появления пары 3 F стек будет содержать пары

$$(0, \Lambda) \qquad (1, \text{A1}) \qquad (3, \text{E3})$$

в направлении снизу вверх.)

A2. [Следующий элемент.] Пусть (L, P) — это следующий элемент данных, взятый из входного потока. После исчерпания входного потока выполнение алгоритма прекращается. Установить Q ⇐ AVAIL (т. е. пусть Q — адрес нового узла, в котором можно разместить следующую позицию таблицы данных).

A3. [Установка связей для символьных имен.] Установить

$$\texttt{PREV(Q)} \leftarrow \texttt{LINK(P)}, \qquad \texttt{LINK(P)} \leftarrow \texttt{Q}, \qquad \texttt{NAME(Q)} \leftarrow \texttt{P}.$$

(Так будут заданы значения для двух связей из пяти в узле NODE(Q). Теперь нужно соответствующим образом установить значения связей PARENT, CHILD и SIB.)

A4. [Сравнение уровней.] Пусть пара (L1, P1) является верхним элементом стека. Если L1 < L, установить CHILD(P1) ← Q (или, если P1 = Λ, установить FIRST ← Q, где FIRST — переменная, которая будет указывать на первый элемент таблицы данных) и перейти к шагу A6.

A5. [Удаление верхнего элемента.] Если L1 > L, то удалить верхний элемент стека. Пусть, например, (L1, P1) — новый элемент, который только что был удален из верхней части стека. Затем повторить шаг A5. Если L1 < L (т. е. на одном уровне обнаружены разные номера), выдать сообщение об ошибке. В противном случае, а именно — при L1 = L, установить SIB(P1) ← Q и удалить верхний элемент стека. Пусть, например, пара (L1, P1) является парой, которая только что была удалена из верхней части стека.

A6. [Установка связей семьи.] Установить PARENT(Q) ← P1, CHILD(Q) ← Λ, SIB(Q) ← Λ.

A7. [Ввести элемент в стек.] Поместить пару (L, Q) в верхнюю часть стека и вернуться к шагу A2. ▮

Благодаря введению вспомогательного стека, который описан на шаге A1, данный алгоритм настолько упрощается, что не требует дополнительных разъяснений.

Следующая задача заключается в поиске позиции таблицы данных, соответствующей ссылке

$$A_0 \ \texttt{OF} \ A_1 \ \texttt{OF} \ \ldots \ \texttt{OF} \ A_n, \qquad n \geq 0. \tag{6}$$

В хорошем компиляторе следует также предусмотреть проверку недвусмысленности такой ссылки. В этом случае сразу же напрашивается следующий алгоритм (рис. 40). Все, что теперь необходимо сделать, — просмотреть список позиций таблицы данных для имени A_0 и убедиться в том, что в точности одна из них соответствует квалификации $A_1, \ldots, A_n$.

Алгоритм B (*Проверка квалифицированной ссылки*). В соответствии со ссылкой (6) программа таблицы символов найдет указатели $P_0, P_1, \ldots, P_n$ на позиции таблицы символов $A_0, A_1, \ldots, A_n$ соответственно.

Назначение данного алгоритма заключается в проверке $P_0, P_1, \ldots, P_n$ и либо определении того, что ссылка (6) ошибочна, либо в установлении для значения переменной Q адреса позиции таблицы данных для элемента, на который ссылается (6).

B1. [Инициализация.] Установить Q ← Λ, P ← LINK(P_0).

B2. [Готово?] Если P = Λ, то выполнение алгоритма прекращается; в этот момент Q равно Λ, если (6) не соответствует никакой позиции таблицы данных. Но если

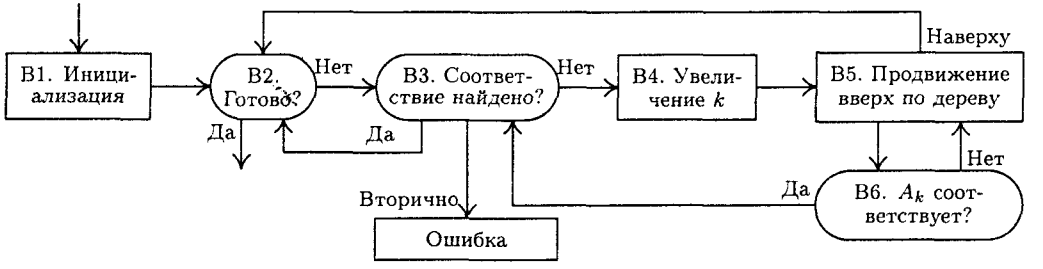


Рис. 40. Алгоритм для проверки ссылок в языке COBOL.

$\texttt{P} \neq \Lambda$, установить $\texttt{S} \leftarrow \texttt{P}$ и $k \leftarrow 0$. ($\texttt{S}$ — переменная-указатель, значения которой меняются от $\texttt{P}$ и ведут вверх по дереву по связям PARENT; k — целочисленная переменная, которая принимает значения от 0 к n. На практике указатели $P_0, \ldots, P_n$ часто содержатся в связанном списке, и тогда вместо k используется переменная-указатель, которая совершает обход этого списка; см. упр. 5.)

B3. [Соответствие найдено?] Если $k < n$, то перейти к шагу B4. В противном случае найдена соответствующая позиция таблицы данных. Если $\texttt{Q} \neq \Lambda$, то найдена вторая такая позиция, и поэтому нужно отослать сообщение об ошибке. Установить $\texttt{Q} \leftarrow \texttt{P}$, $\texttt{P} \leftarrow \texttt{PREV(P)}$ и перейти к шагу B2.

B4. [Увеличение k.] Установить $k \leftarrow k + 1$.

B5. [Продвижение вверх по дереву.] Установить $\texttt{S} \leftarrow \texttt{PARENT(S)}$. Если $\texttt{S} = \Lambda$, то соответствие найти не удалось; установить $\texttt{P} = \texttt{PREV(P)}$ и перейти к шагу B2.

B6. [A_k соответствует?] Если $\texttt{NAME(S)} = P_k$, перейти к шагу B3; в противном случае перейти к шагу B5. ▮

Обратите внимание, что связи CHILD и SIB в этом алгоритме не использовались.

Третий, и последний, из нужных нам алгоритмов имеет отношение к команде MOVE CORRESPONDING. Прежде чем приступить к созданию алгоритма, нужно четко сформулировать его назначение. В языке COBOL выражение

$$\texttt{MOVE CORRESPONDING}\ \alpha\ \texttt{TO}\ \beta, \tag{7}$$

где α и β — ссылки наподобие (6) на элементы данных, обозначает сокращенную запись множества всех выражений типа

$$\texttt{MOVE}\ \alpha'\ \texttt{TO}\ \beta',$$

для которых существует такое целое число $n \geq 0$ и такие n имен $A_0, A_1, \ldots, A_{n-1}$, что

$$\begin{aligned} \alpha' &= A_0\ \texttt{OF}\ A_1\ \texttt{OF}\ \ldots\ \texttt{OF}\ A_{n-1}\ \texttt{OF}\ \alpha, \\ \beta' &= A_0\ \texttt{OF}\ A_1\ \texttt{OF}\ \ldots\ \texttt{OF}\ A_{n-1}\ \texttt{OF}\ \beta \end{aligned} \tag{8}$$

и либо α', либо β' является простейшим элементом (а не группой элементов). Более того, необходимо, чтобы в (8) была указана *полная* квалификация первых уровней, а именно — что A_{j+1} является родителем A_j для $0 \leq j < n$. α' и β' должны располагаться в дереве на n уровней ниже, чем α и β.

Для рассматриваемого здесь примера (4) выражение

```
MOVE CORRESPONDING A TO H
```

является сокращенной формой записи выражений

```
MOVE B OF A TO B OF H
MOVE G OF F OF A TO G OF F OF H
```

Алгоритм для распознавания соответствующих пар α', β' несмотря на свою простоту довольно интересен, т. е. необходимо совершить обход дерева с корнем α в прямом порядке, одновременно выискивая в дереве β совпадающие имена и пропуская поддеревья, в которых появление соответствующих элементов невозможно. Имена $A_0, \ldots, A_{n-1}$ из (8) располагаются в обратном порядке: $A_{n-1}, \ldots, A_0$.

Алгоритм С (*Поиск соответствующих пар*). Для заданных PO и QO, которые указывают на позиции таблицы данных для α и β соответственно, этот алгоритм последовательно находит все пары указателей (P, Q) на элементы (α', β'), удовлетворяющие упомянутым выше требованиям.

С1. [Инициализация.] Установить P $\leftarrow$ PO, Q $\leftarrow$ QO. (В оставшейся части этого алгоритма указательные переменные P и Q совершают обход деревьев с корнями α и β соответственно.)

С2. [Простейший элемент?] Если CHILD(P) $= \Lambda$ или CHILD(Q) $= \Lambda$, то вывести (P, Q) как одну из искомых пар и перейти к шагу С5. В противном случае установить P $\leftarrow$ CHILD(P), Q $\leftarrow$ CHILD(Q). (На этом шаге P и Q указывают на элементы α' и β', удовлетворяющие (8), а команду MOVE α' TO β' необходимо выполнять тогда и только тогда, когда либо α', либо β' (или оба сразу) — простейший элемент.)

С3. [Сравнение имен.] (P и Q указывают на элементы данных, которые соответственно имеют квалификацию в виде

$$A_0 \texttt{ OF } A_1 \texttt{ OF } \ldots \texttt{ OF } A_{n-1} \texttt{ OF } \alpha$$

и

$$B_0 \texttt{ OF } A_1 \texttt{ OF } \ldots \texttt{ OF } A_{n-1} \texttt{ OF } \beta.$$

Теперь задача заключается в том, чтобы узнать, можно ли сделать так, чтобы $B_0 = A_0$, проверяя все имена в группе A_1 OF ... OF A_{n-1} OF β.) Если NAME(P) $=$ NAME(Q), то перейти к шагу С2 (найдено совпадение). В противном случае, если SIB(Q) $\neq \Lambda$, установить Q $\leftarrow$ SIB(Q) и повторить шаг С3. (Если SIB(Q) $= \Lambda$, значит, в этой группе нет соответствующего имени и следует перейти к шагу С4.)

С4. [Продвижение.] Если SIB(P) $\neq \Lambda$, то установить значения P $\leftarrow$ SIB(P), Q $\leftarrow$ CHILD(PARENT(Q)) и вернуться к шагу С3. Если SIB(P) $= \Lambda$, то установить P $\leftarrow$ PARENT(P) и Q $\leftarrow$ PARENT(Q).

С5. [Готово?] Если P $=$ PO, то прекратить выполнение алгоритма; в противном случае перейти к шагу С4. ▮

Блок-схема этого алгоритма показана на рис. 41. Доказательство корректности алгоритма можно легко получить с помощью метода индукции по размеру обрабатываемых деревьев (см. упр. 9).

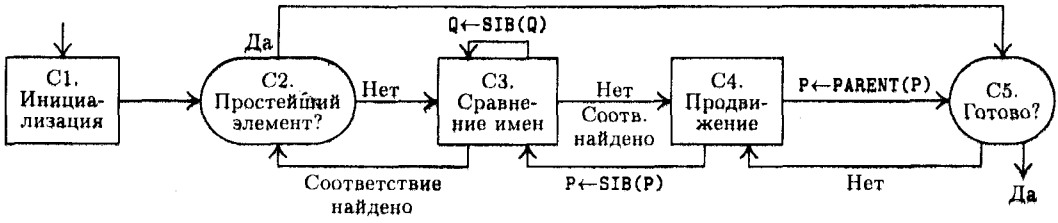


Рис. 41. Алгоритм для выполнения команды MOVE CORRESPONDING.

Теперь рассмотрим способы применения пяти полей связи (PREV, PARENT, NAME, CHILD и SIB) в алгоритмах B и C. Замечательно то, что эти связи образуют "полный набор" в том смысле, что алгоритмы B и C действительно выполняют минимальный объем работы при продвижении по таблице данных. И всякий раз, когда нужно сослаться на другую позицию таблицы данных, ее адрес сразу же становится доступным, поэтому нет необходимости проводить дополнительный поиск. Трудно представить, как можно было бы сделать алгоритмы B и C более быстрыми, включив в таблицу дополнительную информацию о связях (см., однако, упр. 11).

Каждое поле связи может рассматриваться как *ключ* к этой программе, который используется для ускорения работы алгоритмов. (Конечно, алгоритм для построения таблиц, т. е. алгоритм A, выполняется медленнее, поскольку он имеет больше связей, которые необходимо заполнить. Но таблица строится лишь один раз.) С другой стороны, ясно, что в построенной выше таблице данных содержится гораздо больше избыточной информации. Посмотрим, что произойдет, если *удалить* некоторые поля связи.

Связь PREV, хотя она и не используется в алгоритме C, имеет большое значение в алгоритме B и, похоже, является существенной частью любого компилятора языка COBOL, за исключением случаев, когда требуется выполнять длительные операции поиска. Следовательно, поле, которое связывает все элементы с одинаковыми именами, имеет большое значение для эффективной работы. Поэтому стратегию действий в такой ситуации можно слегка модифицировать и вместо связи Λ в конце каждого списка применить циклическое связывание. Но этого не стоит делать, когда поля связи не изменяются и не удаляются.

Связь PARENT используется в алгоритмах B и C, хотя можно обойтись и без нее, если в алгоритме C применить вспомогательный стек или добавить связь SIB так, чтобы задействовать связи-нити (как в разделе 2.3.2). Таким образом, становится ясно, что связь PARENT используется только в алгоритме B. Если бы поле SIB было связью-нитью, чтобы элементы со значением поля связи SIB $= \Lambda$ содержали вместо него значение SIB = PARENT, можно было бы найти родителя любого элемента данных, следуя по связям SIB. Добавленные связи-нити можно отличить либо с помощью нового поля TAG в каждом узле, в котором указывалось бы, что поле SIB содержит связь-нить, либо с помощью условия SIB(P) < P, если позиции таблицы данных хранятся последовательно в памяти в порядке появления. Это значит, что на шаге B5 придется выполнить короткий поиск, а сам алгоритм соответственно станет работать медленнее.

Связь NAME используется в этих алгоритмах только на шагах B6 и C3. В обоих случаях можно было бы выполнить проверку NAME(S) $= P_k$ и NAME(P) = NAME(Q)

иначе, если бы связь NAME отсутствовала (см. упр. 10), но это существенно замедлило бы выполнение внутренних циклов в алгоритмах B и C. Ясно, что здесь снова имеет место компромисс между экономией пространства для связи и скоростью выполнения алгоритмов. (Скорость работы алгоритма C не так уж важна для компиляторов языка COBOL, если рассматривать типичные способы употребления команд MOVE CORRESPONDING. Однако алгоритм B должен работать быстро.) Известно, что связи NAME используются и для выполнения других важных задач в компиляторе языка COBOL, особенно при выводе диагностической информации.

Алгоритм A постепенно создает таблицу данных, причем никогда не приходится возвращать узлы в область свободной памяти. Поэтому позиции таблицы данных обычно располагаются в последовательных ячейках памяти в порядке появления элементов данных в исходной программе на языке COBOL. Таким образом, в нашем примере (5) ячейки A1, B3, ... будут располагаться одна за другой. Такой последовательный характер размещения ячеек в таблице данных позволяет существенно упростить работу. Например, связь CHILD каждого узла либо равна Λ, либо указывает на узел, который следует сразу же за ним, поэтому поле CHILD можно сократить до размера 1 бит. Или поле CHILD можно удалить и вместо него проверить условие PARENT$(P + c) = P$, где c — размер узла в таблице данных.

Таким образом, необязательно использовать сразу все пять полей связи, хотя они очень полезны для ускорения работы алгоритмов B и C. Эта ситуация весьма типична для большинства многосвязных структур.

Интересно отметить, что по крайней мере полдюжины разработчиков компиляторов COBOL в начале 60-х годов независимо пришли к одному способу организации таблицы данных с помощью пяти связей (или четырех из пяти, так как связь CHILD обычно не используется). Впервые описание такого метода было опубликовано Г. В. Лоусоном (мл.) (H. W. Lawson, Jr.) [см. *ACM National Conference Digest* (Syracuse, N.Y.: 1962]. Однако Дэвид Дам (David Dahm) в 1965 году предложил оригинальный метод, который позволяет получить тот же результат, что и при работе с алгоритмами B и C, но с использованием двух полей связи и последовательного распределения позиций в таблице данных без существенного снижения скорости выполнения (см. упр. 12–14).

УПРАЖНЕНИЯ

1. [*00*] Если конфигурации данных в языке COBOL рассматривать как древовидные структуры, то в каком порядке они записаны: в прямом, обратном или ни в одном из них?

2. [*10*] Оцените время выполнения алгоритма A.

3. [*22*] Структуры данных языка PL/I подобны структурам языка COBOL, но в них допустима любая последовательность номеров уровней. Например, последовательность

1 A		1 A
3 B		2 B
5 C	эквивалентна последовательности	3 C
4 D		3 D
2 E		2 E

В итоге правило (a) изменяется таким образом: "Элементы группы должны иметь невозрастающую последовательность номеров уровней, причем все они должны быть больше номера уровня группы данной группы". Как необходимо изменить алгоритм A, чтобы

выполнить переход от соглашений, принятых для языка COBOL, к соглашениям, принятым для языка PL/I?

▸ **4.** [*26*] Алгоритм A не позволит обнаружить ошибку, если программист в COBOL-программе нарушит упомянутое в этом разделе правило (c). Как следует изменить алгоритм A, чтобы принимались только те структуры, которые удовлетворяют правилу (c)?

5. [*20*] На практике алгоритм B может получать из входного потока связанный список ссылок на таблицу символов, а не то, что прежде называлось "$P_0, P_1, \ldots, P_n$". Пусть T является такой указательной переменной, что

$$\texttt{INFO(T)} \equiv P_0,\ \texttt{INFO(RLINK(T))} \equiv P_1,\ \ldots,\ \texttt{INFO(RLINK}^{[n]}\texttt{(T))} \equiv P_n,\ \texttt{RLINK}^{[n+1]}\texttt{(T)} = \Lambda.$$

Покажите, как изменить алгоритм B, чтобы в нем в качестве входного потока можно было использовать связанный список.

6. [*23*] В языке PL/I допускается использование структур данных, которые во многом подобны структурам языка COBOL, но без применения ограничения (c). Вместо этого получим правило, по которому квалифицированная ссылка (3) является однозначной, если указана "полная" квалификация, т. е. если A_{j+1} — родитель A_j для $0 \le j < n$ и если A_n не имеет родителя. Ограничение (c) теперь сведено к простому условию, согласно которому никакие два элемента данных группы не могут иметь одно и то же имя. На второе появление CC в (2) можно было бы недвусмысленно сослаться с помощью ссылки "CC OF AA", а на три элемента данных

```
1 A
  2 A
    3 A
```

можно было бы в соответствии с соглашениями, принятыми в языке PL/I, сослаться в такой форме: "A", "A OF A", "A OF A OF A". [*Замечание.* На самом деле "OF" заменяется точкой в языке PL/I, а порядок является реверсивным. Так, вместо "CC OF AA" в языке PL/I используется обозначение "AA.CC", но для данного упражнения это несущественно.] Покажите, как можно модифицировать алгоритм B, чтобы он удовлетворял соглашениям, принятым для языка PL/I.

7. [*15*] Что означает в языке COBOL команда MOVE CORRESPONDING SALES TO PURCHASES по отношению к структуре данных (1)?

8. [*10*] При каких условиях команда MOVE CORRESPONDING α TO β означает то же самое, что и команда MOVE α TO β, в соответствии с данным в этом разделе определением?

9. [*M23*] Докажите корректность алгоритма C.

10. [*23*] (a) Как можно было бы выполнить проверку условия NAME(S) $= P_k$ на шаге B6, если бы в узлах таблицы данных не было связи NAME? (b) Как можно было бы выполнить проверку NAME(P) = NAME(Q) на шаге C3, если бы в узлах таблицы данных не было связи NAME? (Предположим, что все другие связи присутствуют, как описано в этом разделе.)

▸ **11.** [*23*] Какие дополнительные связи или изменения в стратегии создания алгоритмов могли бы ускорить работу алгоритма B или C?

12. [*25*] (Задача Д. М. Дама (D. M. Dahm).) Рассмотрим представление таблицы данных в последовательных ячейках памяти с помощью всего двух связей для каждого элемента данных:

PREV (так же, как в данном разделе);

SCOPE (связь с последним простейшим элементом в данной группе).

Получим SCOPE(P) = P тогда и только тогда, когда NODE(P) представляет собой простейший элемент. Например, таблица данных (5) будет заменена таким представлением.

	PREV	SCOPE		PREV	SCOPE		PREV	SCOPE
A1:	Λ	G4	F3:	Λ	G4	B5:	B3	B5
B3:	Λ	D7	G4:	Λ	G4	C5:	C7	G9
C7:	Λ	C7	H1:	Λ	G9	E9:	E3	E9
D7:	Λ	D7	F5:	F3	G8	D9:	D7	D9
E3:	Λ	E3	G8:	G4	G8	G9:	G8	G9

(Ср. с (5) из раздела 2.3.3.) Обратите внимание, что NODE(P) является частью дерева под узлом NODE(Q) тогда и только тогда, когда $\texttt{Q} < \texttt{P} \le \texttt{SCOPE(Q)}$. Создайте алгоритм, который выполняет функции алгоритма B, если таблица данных имеет такой формат.

▶ **13.** [*24*] Предложите вместо алгоритма A другой алгоритм для работы с таблицей данных с форматом, описанным в упр. 12.

▶ **14.** [*28*] Предложите вместо алгоритма C другой алгоритм для работы с таблицей данных с форматом, описанным в упр. 12.

15. [*25*] (Задача Дэвида С. Вайса (David S. Wise).) Измените алгоритм A так, чтобы для стека не использовалось никакое дополнительное пространство. [*Указание.* Поля SIB всех узлов с указателями в стеке в этой формулировке равны Λ.]

2.5. ДИНАМИЧЕСКОЕ ВЫДЕЛЕНИЕ ПАМЯТИ

В ПРЕДЫДУЩИХ РАЗДЕЛАХ было показано, как использование связей приводит к тому, что структуры данных могут располагаться в памяти непоследовательно; несколько таблиц могут независимо расти и уменьшаться в области общего пула памяти. Однако мы всегда молчаливо предполагали, что узлы имеют один и тот же размер, т. е. каждый узел занимает некоторое фиксированное количество ячеек памяти.

Для очень многих приложений можно найти компромиссное решение, при котором в действительности используется один размер узла для всех структур (например, см. упр. 2). Вместо максимального размера (и потери памяти в малых узлах) зачастую выбирается меньший размер узла и применяется метод, который можно назвать классической *философией связанной памяти*: "Если для размещения информации в одном месте не хватает памяти, разместим ее в другом месте и установим с ней связь".

Однако для очень большого количества приложений использовать единый размер узлов неразумно, ведь часто необходимы узлы различных размеров, разделяющие общую область памяти, т. е. нужны алгоритмы для резервирования и освобождения блоков переменного размера в большой области памяти, причем эти блоки должны состоять из последовательных ячеек памяти. Такие технологии, в целом, называются алгоритмами *динамического выделения памяти*.

Зачастую в моделирующих программах требуется динамическое выделение памяти для узлов весьма малого размера (скажем, от одного до десяти слов). В других случаях, чаще всего — в операционных системах, мы, в первую очередь, работаем с довольно большими блоками информации. Такие "точки зрения" приводят к несколько отличающимся подходам к динамическому выделению памяти, хотя в этих методах много общего. Чтобы унифицировать терминологию рассматриваемых подходов, в данном разделе для обозначения множества последовательных ячеек памяти вместо термина *узел* будем использовать термины *блок* и *область*.

Некоторые авторы начиная примерно с 1975 года называют пул доступной памяти кучей (heap), однако в настоящем издании этот термин используется только в более традиционном смысле, связанном с приоритетными очередями (см. раздел 5.2.3).

А. Резервирование. На рис. 42 представлена типичная *карта памяти*, или "шахматная доска", — диаграмма, отображающая текущее состояние некоторого пула памяти. В приведенном примере она разбита на 53 блока, которые "зарезервированы" (reserved), т. е. используются вперемешку с 21 "свободным" (free) или "доступным" (available) блоком, который не используется. Память компьютера спустя некоторое время работы системы динамического выделения, вероятно, будет выглядеть примерно так. Наша первая задача состоит в поиске ответов на два вопроса.

a) Как можно представить такое разбиение свободной памяти в компьютере?
b) Какой алгоритм при данном распределении свободной памяти достаточно хорош для поиска блока из n последовательных свободных ячеек и его резервирования?

Ответ на вопрос (а) кроется, конечно, в содержании *списка* свободной памяти в некотором месте; почти всегда для этой цели лучше всего использовать ту самую свободную память, сведения о которой содержатся в списке (исключением из этого

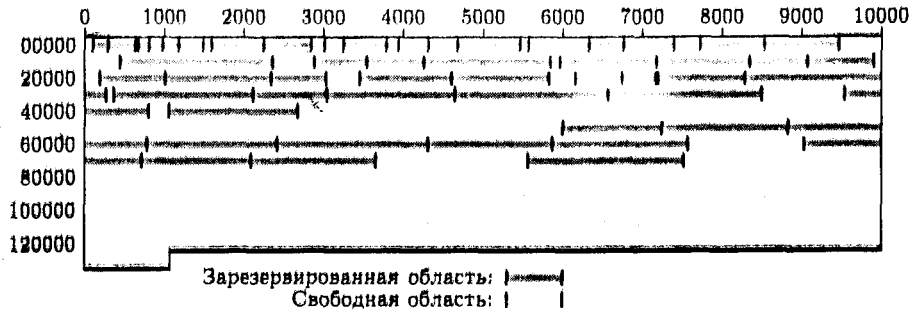


Рис. 42. Карта памяти.

правила может быть дисковая или другая память с различным временем доступа; в таком случае лучше иметь каталог доступного пространства отдельно).

Следовательно, можно *связать вместе* доступные сегменты: первое слово каждой свободной области памяти может содержать размер этого блока и адрес следующей свободной области. Свободные блоки могут быть связаны в порядке возрастания или убывания по размерам, адресам памяти либо в произвольном порядке.

Рассмотрим, например, рис. 42, на котором иллюстрируется состояние памяти объемом 131 072 слова, адресуемых от 0 до 131 071. Чтобы связать свободные блоки в порядке их адресов, потребуется переменная AVAIL, указывающая на первый свободный блок (в нашем случае AVAIL равна 0); другие же блоки будут представлены следующим образом.

Адрес	SIZE	LINK	
0	101	632	
632	42	1488	
⋮	⋮	⋮	[17 подобных записей]
73654	1909	77519	
77519	53553	Λ	[Специальный маркер для последней связи]

Следовательно, ячейки 0–100 образуют первый свободный блок; после занятых областей в ячейках 101–290 и 291–631, показанных на рис. 42, имеется свободное пространство с адресами 632–673; и т. д.

По поводу вопроса (b) понятно, что, если необходима область размером n последовательных слов, нужно выполнить поиск некоторого блока из $m \geq n$ доступных слов и уменьшить его размер до $m - n$. (Кроме того, при $m = n$ следует удалить данный блок из списка свободных.) В наличии может быть несколько блоков размером n или более ячеек, а потому один вопрос превращается в другой: "*Какая именно* область должна быть выделена?".

Два основных ответа на этот вопрос напрашиваются сами собой: можно использовать *метод наилучшего подходящего* или *метод первого подходящего*. В первом случае мы выбираем область с m ячейками памяти, где m — наименьшее значение из имеющихся, не меньшее n. Для такого выбора может потребоваться поиск по всему списку свободного пространства. Метод первого подходящего, с другой стороны, просто выбирает первую попавшуюся область размером не менее n слов.

Исторически чаще использовался метод наилучшего подходящего, так как более разумным представляется подход, сохраняющий бóльшие свободные области "на потом", когда они могут понадобиться. Однако имеется ряд претензий к этому методу: он достаточно медленный, поскольку требует длинного полного поиска, и если он не намного лучше метода первого подходящего в других отношениях, то временем выделения памяти при оценке метода пренебречь нельзя. Еще более важно то, что метод лучшего подходящего имеет тенденцию к увеличению количества блоков свободной памяти малого размера, что обычно нежелательно. Существует ряд ситуаций, в которых метод первого подходящего превосходит метод наилучшего подходящего. Так, например, предположим, что есть только две свободные области памяти размером 1300 и 1200 и три последовательных запроса на выделение блоков памяти размером 1000, 1100 и 250.

Запрос блока размером	*Доступные области, метод первого подходящего*	*Доступные области, метод наилучшего подходящего*
—	1300, 1200	1300, 1200
1000	300, 1200	1300, 200
1100	300, 100	200, 200
250	50, 100	Нужного блока нет

(1)

(Противоположный пример приведен в упр. 7.) Поскольку ни один метод явно не превосходит другой, можно порекомендовать метод первого подходящего*.

Алгоритм А (*Метод первого подходящего*). Пусть AVAIL указывает на первый доступный блок памяти, и предположим, что каждый свободный блок с адресом P имеет два поля: SIZE(P) — количество слов в блоке и LINK(P) — указатель на следующий свободный блок. Последний указатель равен Λ (что указывает на завершение списка блоков свободной памяти). Алгоритм находит и выделяет блок размером N слов (или сообщает о невозможности выделения запрошенной памяти).

A1. [Инициализация.] Установить Q $\leftarrow$ LOC(AVAIL). (Везде в алгоритме используются два указателя, Q и P, которые, вообще говоря, связаны соотношением P = LINK(Q). Мы полагаем, что LINK(LOC(AVAIL)) = AVAIL.)

A2. [Конец списка?] Установить P $\leftarrow$ LINK(Q). Если P = Λ, алгоритм завершается неудачно и блок размером N последовательных слов не может быть выделен.

A3. [Достаточен ли размер блока?] Если SIZE(P) $\geq$ N, перейти к шагу A4; в противном случае установить Q $\leftarrow$ P и вернуться к шагу A2.

A4. [Выделение блока.] Установить K $\leftarrow$ SIZE(P) $-$ N. Если K = 0, установить LINK(Q) $\leftarrow$ LINK(P) (тем самым удаляя пустую область из списка); в противном случае установить SIZE(P) $\leftarrow$ K. Алгоритм успешно завершается, выделяя область памяти длиной N, которая начинается с адреса P + K. ▮

---

* Видимо, именно из-за отсутствия явного превосходства какого-либо метода над другим программисту во времена DOS предоставлялось право (хотя и не рекомендовалось) изменять стратегию выделения памяти операционной системой при помощи функции 58h прерывания 21h путем выбора метода первого подходящего, лучшего подходящего или последнего подходящего (включая в более поздних версиях указание на возможность использования верхней (high) памяти). *Прим. перев.*

Данный алгоритм, определенно, несколько прямолинеен. Однако всего лишь небольшое изменение стратегии может существенно повысить скорость его работы. Это весьма важное улучшение алгоритма, и читатель получит удовольствие, выполнив его поиск самостоятельно (см. упр. 6).

Алгоритм A может использоваться как для больших, так и для малых значений N. Временно предположим, однако, что нас интересуют, в первую очередь, *большие* значения N. Рассмотрим, что случится, если в алгоритме SIZE(P) равно N + 1: перейдем к шагу A4 и уменьшим SIZE(P) до 1. Другими словами, будет создан блок доступной памяти размером 1. Он настолько мал, что практически бесполезен и только засоряет систему. Было бы лучше выделить весь блок размером N + 1 слов вместо экономии одного слова; зачастую стóит заплатить несколькими словами памяти за избавление от несущественных деталей. Подобные примечания относятся и к блокам памяти размером N + K слов при очень малом K.

Если допустить выделение блоков немного большего размера, чем N слов, придется помнить размер выделенного блока, чтобы при его освобождении корректно вернуть в пул свободной памяти все N+K слов. Это увеличивает накладные расходы, ведь придется использовать пространство в *каждом* блоке для того, чтобы сделать систему более эффективной в тех случаях, когда имеется почти подходящий блок. Так что подобная модифицированная стратегия не кажется очень привлекательной. Однако зачастую использование в начале каждого блока переменного размера специального *управляющего слова* представляется желательным по множеству других причин, так что предположение о том, что в первом слове каждого блока, как выделенного, так и свободного, присутствует поле SIZE, вполне разумно.

В соответствии с этими соглашениями можно изменить шаг A4 приведенного выше алгоритма следующим образом.

A4′. [Выделение $\geq$ N.] Установить K $\leftarrow$ SIZE(P) $-$ N. Если K $< c$ (где c — малая положительная константа, зависящая от того, каким количеством памяти мы готовы пожертвовать для ускорения работы), установить LINK(Q) $\leftarrow$ LINK(P) и L $\leftarrow$ P. В противном случае установить SIZE(P) $\leftarrow$ K, L $\leftarrow$ P + K, SIZE(L) $\leftarrow$ N. Алгоритм успешно завершается, выделив область длиной N или больше, которая начинается с адреса L.

Обычно значение константы c выбирается равным 8 или 10, хотя практически нет никаких теоретических или эмпирических оснований для сравнения этих значений с другими. При использовании метода наилучшего подходящего проверка K $< c$ *более важна*, чем в случае первого подходящего, поскольку компактное размещение (меньшие значения K) встречается при таком методе более часто, а число свободных блоков в этом алгоритме должно быть как можно меньше.

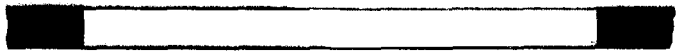
B. Освобождение. Теперь рассмотрим обратную задачу. Каким образом вернуть блоки в список свободного пространства, когда необходимость в них отпадает?

Пожалуй, заманчиво было бы избежать решения данной задачи, используя сборку мусора* (см. раздел 2.3.5). Следуя этой политике, мы просто ничего не делаем до тех пор, пока пространство не окажется заполненным, а затем ищем все используемые в текущий момент области памяти и строим новый список AVAIL.

* Именно этот метод работы с освобождаемой памятью используется, например, в языке *Java*. — *Прим. перев.*

Однако идея сборки мусора не рекомендуется для всех приложений. На первое место выходит вопрос строгой дисциплины использования указателей, если нужно гарантировать простое нахождение всех используемых в настоящий момент областей памяти, а именно этой дисциплины зачастую и не хватает рассматриваемым здесь приложениям. Во-вторых, как мы уже видели, метод сборки мусора имеет тенденцию к замедлению работы при почти заполненной памяти.

Есть и еще одна, более важная (хотя и не встречавшаяся до сих пор при обсуждении этого метода) причина, по которой сборка мусора нас не устраивает. Предположим, что имеются две соседние свободные области памяти, но из-за использования технологии сборки мусора одна из них (заштрихованная) пока не попала в список AVAIL.



(2)

На этой диаграмме черные области памяти слева и справа зарезервированы и недоступны. По запросу можно зарезервировать часть области памяти, о которой *известно*, что она свободна:

. (3)

Если в этот момент произойдет сборка мусора, получатся две раздельные свободные области памяти:

. (4)

Границы между свободной и выделенной памятью имеют тенденцию к самовоспроизводству, и со временем ситуация становится все хуже и хуже. Но если бы использовалась политика немедленного возврата освобожденных блоков памяти в список AVAIL и *слияния смежных свободных блоков*, то (2) тут же превратилась бы в

(5)

и при выделении памяти получилось бы такое распределение блоков памяти

, (6)

которое гораздо лучше, чем (4). Как видите, рассмотренное явление вызывает большее дробление памяти при использовании метода сборки мусора, чем должно быть.

Для устранения этой проблемы можно использовать сборку мусора вместе с процессом *уплотнения памяти*, т. е. перемещения всех выделенных блоков в соседние позиции, чтобы после сборки мусора все свободные блоки были объединены. Алгоритм выделения памяти в этой ситуации становится в противоположность алгоритму A тривиальным, поскольку в любой момент есть только один свободный блок. Хотя такой подход требует времени на перемещение всех задействованных блоков и изменение в них значений связей, метод может применяться с достаточной эффективностью при условии дисциплины использования указателей и наличии запасного поля связи в каждом блоке, используемом алгоритмом сборки мусора* (см. упр. 33).

* Здесь следует отметить, что такой метод применим далеко не во всех языках программирования. При перемещении блока памяти должны быть корректно обновлены все указатели на

Поскольку многие приложения не соответствуют этим требованиям, выдвигаемым методом сборки мусора, изучим методы возврата блоков памяти в список свободного пространства. Единственная сложность в этих методах заключается в объединении соседних свободных блоков памяти. Так, когда освобождается блок, находящийся между двумя свободными блоками, все три области памяти должны быть слиты в одну. *Таким образом достигается хорошее равновесие памяти даже при длительном непрерывном процессе выделения и освобождения областей памяти.* (Для доказательства этого факта обратитесь к правилу "50%", приведенному ниже.)

В данном случае задача заключается в определении, является ли соседняя (с той или другой стороны) область памяти свободной, и если она свободна, то необходимо корректно обновить список AVAIL. Последняя операция несколько сложнее, чем кажется из ее названия.

Первое решение этой проблемы состоит в содержании списка AVAIL в порядке возрастания адресов памяти.

Алгоритм B (*Освобождение в рассортированном списке*). В предположениях алгоритма A с дополнительным предположением об упорядоченности списка AVAIL по адресам памяти (т. е. если P указывает на свободный блок и LINK(P) $\neq \Lambda$, то LINK(P) > P) этот алгоритм добавляет блок из N последовательных ячеек, начинающийся с адреса P0, в список AVAIL. Естественно, предполагается, что эти N ячеек уже свободны.

B1. [Инициализация.] Установить Q $\leftarrow$ LOC(AVAIL). (См. примечание к шагу A1.)

B2. [Продвижение P.] Установить P $\leftarrow$ LINK(Q). Если P $= \Lambda$ или если P > P0, перейти к шагу B3; в противном случае установить Q $\leftarrow$ P и повторить шаг B2.

B3. [Проверка верхней границы.] Если P0 + N $=$ P и P $\neq \Lambda$, установить N $\leftarrow$ N + SIZE(P) и установить LINK(P0) $\leftarrow$ LINK(P). В противном случае установить LINK(P0) $\leftarrow$ P.

B4. [Проверка нижней границы.] Если Q + SIZE(Q) = P0 (предполагается, что

$$\text{SIZE(LOC(AVAIL))} = 0,$$

так что условие всегда не выполняется при Q = LOC(AVAIL)), установить SIZE(Q) $\leftarrow$ SIZE(Q) + N и LINK(Q) $\leftarrow$ LINK(P0). В противном случае установить LINK(Q) $\leftarrow$ P0, SIZE(P0) $\leftarrow$ N. ∎

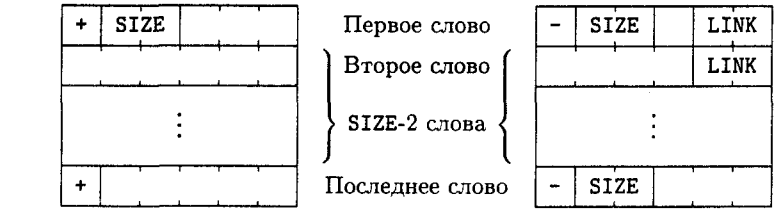
На шагах B3 и B4 выполняется требуемое слияние; учитывается тот факт, что указатели Q < P0 < P являются начальными адресами трех последовательных свободных блоков.

этот блок (и внутрь него). Например, в случае *Java* такой метод может применяться, поскольку в языке отсутствует понятие указателя и реальная адресация блоков памяти переменными может быть отслежена внутренними средствами языка. В то же время в языках наподобие *C* и *Pascal*, в которых используются указатели, отследить создание указателя на ту или иную область памяти *невозможно*, а значит, *применение метода ограничено его использованием только в менеджерах* памяти программ, созданных специальным образом с учетом требований такого менеджера (с дополнительными средствами регистрации всех указателей на выделяемые блоки памяти). (Вопросы косвенной адресации и защищенного режима процессоров не упоминаем, так как они не связаны с тематикой книги.) — *Прим. перев.*

Если список AVAIL не упорядочен, нетрудно понять, что попытка слияния блоков списка "в лоб" приведет к просмотру всего списка AVAIL; алгоритм B в среднем уменьшает этот поиск до примерно *половины* списка AVAIL (на шаге B2). В упр. 11 показано, каким образом можно модифицировать алгоритм B, чтобы в среднем потребовалось просмотреть около одной трети списка AVAIL. Но ясно, что, когда список AVAIL длинный, все эти методы работают существенно медленнее, чем хотелось бы. Нет ли еще какого-либо способа выделения и освобождения памяти, при котором не требовался бы столь пространный поиск в списке AVAIL?

Рассмотрим метод, который позволяет устранить любой поиск при освобождении памяти и может быть модифицирован (см. упр. 6) для резкого сокращения времени поиска при выделении памяти. В этой технологии используется поле TAG в начале и в конце блока, а также поле SIZE в первом слове каждого блока. Такие накладные расходы не столь значительны при использовании больших блоков памяти, хотя, возможно, цена окажется слишком большой при наличии множества блоков очень малого среднего размера. Другой метод, описанный в упр. 19, требует только одного бита в первом слове каждого блока. Ценой этой экономии является несколько большее время работы и усложнение программы*.

Как бы там ни было, положим, что нет возражений против добавления некоторого количества битов управляющей информации для сохранения высокой скорости работы по сравнению с алгоритмом B при длинном списке AVAIL. В описываемом методе предполагается, что каждый блок имеет следующий вид.



(7)

Идея следующего алгоритма состоит в поддержании двусвязного списка AVAIL, так что элементы списка могут быть легко удалены из произвольной части списка. Поле TAG с обоих концов блока можно использовать для управления процессом слияния, поскольку оно позволяет легко определить, свободен ли смежный блок памяти.

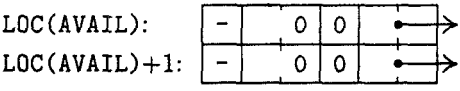
Двойное связывание достигается обычным путем: LINK в первом слове указывает на следующий доступный блок в списке, а LINK во втором слове указывает на предыдущий доступный блок. Таким образом, если P — адрес блока, то

$$\texttt{LINK(LINK(P)} + 1) = \texttt{P} = \texttt{LINK(LINK(P} + 1)). \tag{8}$$

---

* Этот принцип положен, в частности, в основу управления памятью в MS DOS, где каждый блок памяти (MCB — Memory Control Block) имеет поля размера, владельца и типа блока. Менеджер памяти, основанный на списке блоков, использовался, например, в *Turbo Pascal*. — *Прим. перев.*

Для корректности "граничных условий" заголовок списка устанавливается следующим образом.



(9)

Алгоритм для выделения первого подходящего блока очень похож на алгоритм A, и потому здесь не рассматривается (см. упр. 12). Принципиально новое свойство этого метода заключается в освобождении блока за, по сути, фиксированное время.

Алгоритм C (*Освобождение с дескрипторами границ*) Предположим, что блоки памяти выглядят так, как в (7), а список AVAIL имеет две связи, как описывалось выше. Данный алгоритм помещает блок памяти с начальным адресом P0 в список AVAIL. Если пул доступной памяти располагается от адреса m_0 до m_1 включительно, в алгоритме для удобства предполагается, что

$$\texttt{TAG}(m_0 - 1) = \texttt{TAG}(m_1 + 1) = \text{“+”}.$$

C1. [Проверка нижней границы.] Если TAG(P0 − 1) = “+”, перейти к шагу C3.

C2. [Удаление нижней области.] Установить P ← P0 − SIZE(P0 − 1), а затем установить P1 ← LINK(P), P2 ← LINK(P + 1), LINK(P1 + 1) ← P2, LINK(P2) ← P1, SIZE(P) ← SIZE(P) + SIZE(P0), P0 ← P.

C3. [Проверка верхней границы.] Установить P←P0+SIZE(P0). Если TAG(P) =“+”, перейти к шагу C5.

C4. [Удаление верхней границы.] Установить P1 ← LINK(P), P2 ← LINK(P+1), LINK(P1+1) ← P2, LINK(P2) ← P1, SIZE(P0) ← SIZE(P0) + SIZE(P), P ← P + SIZE(P).

C5. [Добавление в список AVAIL.] Установить SIZE(P − 1) ← SIZE(P0), LINK(P0) ← AVAIL, LINK(P0 + 1) ← LOC(AVAIL), LINK(AVAIL + 1) ← P0, AVAIL ← P0, TAG(P0) ← TAG(P − 1) ← “−”. ▮

Шаги алгоритма C определяются видом блоков памяти (7); немного более длинный, но одновременно более быстрый алгоритм можно найти в упр. 15. На шаге C5 AVAIL означает аббревиатуру для LINK(LOC(AVAIL)), как показано в (9).

C. “Система двойников”. Изучим теперь другой подход к динамическому выделению памяти для использования на двоичных компьютерах. В этом методе применяется по одному дополнительному биту на каждый блок; кроме того, все блоки должны иметь длину 1, 2, 4, 8, 16 и т. д. Если длина блока не равна 2^k слов для некоторого целого k, выбирается следующая более высокая степень 2 и часть выделенной памяти при этом не используется.

Суть этого метода заключается в организации отдельных списков доступных блоков каждого размера 2^k, $0 \le k \le m$. Весь пул распределяемого пространства памяти состоит из 2^m слов, адреса которых, предположим, находятся в диапазоне от 0 до $2^m - 1$. Изначально весь блок из 2^m слов свободен. Позже, при требовании блока из 2^k слов и отсутствии свободного блока такого размера больший доступный блок *разбивается* (*split*) на две равные части; в конечном итоге появится блок

необходимого размера 2^k. Когда один блок разделяется на два (каждый половинного размера по сравнению с исходным), эти блоки называются *двойниками* (*buddies**). Позже, когда оба двойника вновь становятся свободны, они объединяются в один большой блок. Таким образом, процесс выделения и освобождения может осуществляться бесконечно, если только в какой-то момент вся доступная память не окажется занятой.

Ключевой факт, лежащий в основе практической ценности этого метода, состоит в том, что если известен адрес блока (адрес первого слова в блоке) и его размер, то известен и адрес его двойника. Например, двойником блока размером 16 с двоичным адресом 101110010110000 является блок с двоичным адресом 101110010100000. Для того чтобы понять, почему это так, сначала заметим, что во время работы алгоритма *адрес блока размером* 2^k *кратен* 2^k. Другими словами, адрес в двоичной записи содержит справа как минимум k нулей. Данное наблюдение легко выводится по индукции: если это верно для всех блоков размером 2^{k+1}, то это, несомненно, справедливо и при делении блока пополам.

Значит, блок размером, скажем, 32 имеет адрес вида $xx\ldots x00000$ (где иксы представляют собой 0 или 1); при разделении блока вновь образуемые блоки-двойники имеют адреса $xx\ldots x00000$ и $xx\ldots x10000$. В общем, пусть

$$\text{двойник}_k(x)$$

обозначает адрес двойника блока размером 2^k, адрес которого равен x. Тогда находим, что

$$\text{двойник}_k(x) = \begin{cases} x + 2^k, & \text{если } x \bmod 2^{k+1} = 0; \\ x - 2^k, & \text{если } x \bmod 2^{k+1} = 2^k. \end{cases} \tag{10}$$

Эта функция легко вычисляется с помощью операции исключающее или (иногда называемой селективным дополнением или сложением без переноса), обычно имеющейся на двоичном компьютере (см. упр. 28).

Система двойников использует однобитовое поле TAG в каждом блоке:

$$\begin{aligned} &\texttt{TAG(P)} = 0, && \text{если блок с адресом P выделен;} \\ &\texttt{TAG(P)} = 1, && \text{если блок с адресом P свободен.} \end{aligned} \tag{11}$$

Кроме поля TAG, имеющегося в каждом блоке, в *свободных* блоках есть два поля связи, LINKF и LINKB, которые представляют обычные связи вперед и назад в двусвязном списке; также имеется поле KVAL, определяющее k для блока размером 2^k. В приведенном ниже алгоритме используются ячейки таблицы AVAIL[0], AVAIL[1], ..., AVAIL[m], которые служат соответственно в качестве заголовков списков свободной памяти размером 1, 2, 4, ..., 2^m. Это списки с двойными связями, так что, как обычно, заголовок списка содержит два указателя (см. раздел 2.2.5):

$$\begin{aligned} &\texttt{AVAILF}[k] = \texttt{LINKF(LOC(AVAIL}[k]\texttt{))} = \text{связь с окончанием списка AVAIL}[k]; \\ &\texttt{AVAILB}[k] = \texttt{LINKB(LOC(AVAIL}[k]\texttt{))} = \text{связь с началом списка AVAIL}[k]. \end{aligned} \tag{12}$$

Изначально перед выделением памяти мы имеем

$$\begin{aligned} \texttt{AVAILF}[m] &= \texttt{AVAILB}[m] = 0, \\ \texttt{LINKF(0)} &= \texttt{LINKB(0)} = \texttt{LOC(AVAIL}[m]\texttt{)}, \\ \texttt{TAG(0)} &= 1, \qquad \texttt{KVAL(0)} = m \end{aligned} \tag{13}$$

* Дословный перевод слова *buddy* — *дружище, приятель*. — *Прим. перев.*

(что указывает на единственный свободный блок длиной 2^m, начинающийся по адресу 0) и

$$\texttt{AVAILF}[k] = \texttt{AVAILB}[k] = \texttt{LOC}(\texttt{AVAIL}[k]) \qquad \text{для } 0 \le k < m \tag{14}$$

(что указывает на пустые списки свободных блоков размером 2^k для всех $k < m$).

Исходя из этого описания системы двойников, читатель может самостоятельно и не без определенного удовольствия разработать необходимые алгоритмы для выделения и освобождения областей памяти, прежде чем знакомиться с приведенными далее алгоритмами. Обратите внимание на сравнительную простоту, с которой каждый блок может быть разделен пополам в алгоритме для выделения памяти.

Алгоритм R (*Выделение памяти в системе двойников*). Этот алгоритм предназначен для поиска и выделения блока памяти размером 2^k (или сообщения о невозможности такого выделения) с помощью описанной выше системы двойников.

R1. [Поиск блока.] Пусть j — наименьшее целое число в диапазоне $k \le j \le m$, для которого $\texttt{AVAILF}[j] \neq \texttt{LOC}(\texttt{AVAIL}[j])$, т. е. для которого список свободных блоков размером 2^j не пуст. Если такого j не существует, алгоритм завершается неудачей, поскольку нет ни одного блока достаточного размера для выделения запрошенного количества памяти.

R2. [Удаление из списка.] Установить $\texttt{L} \leftarrow \texttt{AVAILF}[j]$, $\texttt{P} \leftarrow \texttt{LINKF}(\texttt{L})$, $\texttt{AVAILF}[j] \leftarrow \texttt{P}$, $\texttt{LINKB}(\texttt{P}) \leftarrow \texttt{LOC}(\texttt{AVAIL}[j])$ и $\texttt{TAG}(\texttt{L}) \leftarrow 0$.

R3. [Требуется разделение?] Если $j = k$, алгоритм завершается (найден и выделен свободный блок, начинающийся с адреса L).

R4. [Разделение.] Уменьшить j на 1. Затем установить $\texttt{P} \leftarrow \texttt{L} + 2^j$, $\texttt{TAG}(\texttt{P}) \leftarrow 1$, $\texttt{KVAL}(\texttt{P}) \leftarrow j$, $\texttt{LINKF}(\texttt{P}) \leftarrow \texttt{LINKB}(\texttt{P}) \leftarrow \texttt{LOC}(\texttt{AVAIL}[j])$, $\texttt{AVAILF}[j] \leftarrow \texttt{AVAILB}[j] \leftarrow \texttt{P}$. (Тем самым разделяется большой блок памяти и неиспользуемая половина вносится в список $\texttt{AVAIL}[j]$, который был пуст.) Вернуться к шагу R3. ▮

Алгоритм S (*Освобождение памяти в системе двойников*). Этот алгоритм предназначен для возврата блока размером 2^k, начинающегося с адреса L, в область свободной памяти с использованием описанной выше системы двойников.

S1. [Свободен ли двойник?] Установить $\texttt{P} \leftarrow$ двойник$_k$(L) (см. (10)). Если $k = m$ или $\texttt{TAG}(\texttt{P}) = 0$, либо если $\texttt{TAG}(\texttt{P}) = 1$ и $\texttt{KVAL}(\texttt{P}) \neq k$, перейти к шагу S3.

S2. [Объединение двойников.] Установить

$$\texttt{LINKF}(\texttt{LINKB}(\texttt{P})) \leftarrow \texttt{LINKF}(\texttt{P}), \qquad \texttt{LINKB}(\texttt{LINKF}(\texttt{P})) \leftarrow \texttt{LINKB}(\texttt{P}).$$

(Таким образом блок P удаляется из списка $\texttt{AVAIL}[k]$.) Затем установить $k \leftarrow k + 1$ и, если $\texttt{P} < \texttt{L}$, установить $\texttt{L} \leftarrow \texttt{P}$. Вернуться к шагу S1.

S3. [Размещение в списке.] Установить $\texttt{TAG}(\texttt{L}) \leftarrow 1$, $\texttt{P} \leftarrow \texttt{AVAILF}[k]$, $\texttt{LINKF}(\texttt{L}) \leftarrow \texttt{P}$, $\texttt{LINKB}(\texttt{P}) \leftarrow \texttt{L}$, $\texttt{KVAL}(\texttt{L}) \leftarrow k$, $\texttt{LINKB}(\texttt{L}) \leftarrow \texttt{LOC}(\texttt{AVAIL}[k])$, $\texttt{AVAILF}[k] \leftarrow \texttt{L}$. (Таким образом блок L помещается в список $\texttt{AVAIL}[k]$.) ▮

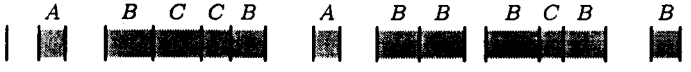
D. Сравнение методов. Выполнение математического анализа этих алгоритмов динамического выделения памяти оказывается весьма трудной задачей, однако

имеется одно интересное явление, которое легко проанализировать, а именно — правило "50%".

Если алгоритмы A и B непрерывно используются таким образом, что система стремится к равновесию, при котором в ней имеется в среднем N выделенных блоков, каждый из которых выделяется и освобождается независимо от других, а величина K в алгоритме A принимает ненулевое значение (или, более того, значения $\geq c$, как на шаге A4′) с вероятностью p, то среднее количество свободных блоков стремится приблизительно к $\frac{1}{2}pN$.

Это правило говорит о том, какой будет приблизительная длина списка AVAIL. Когда величина p близка к 1, что случается при очень малых c и если размеры блоков редко равны, количество свободных блоков составляет примерно половину от количества занятых; отсюда и происходит название правила "50%".

Данное правило нетрудно доказать. Рассмотрим следующую карту памяти:



На ней показаны выделенные блоки памяти трех категорий:

A: при освобождении блока количество свободных блоков уменьшается на единицу;
B: при освобождении блока количество свободных блоков не изменяется;
C: при освобождении блока количество свободных блоков увеличивается на единицу.

Теперь пусть N — число выделенных, а M — число свободных блоков. Пусть A, B и C — количество блоков описанных выше типов. Имеем

$$\begin{aligned} N &= A + B + C; \\ M &= \tfrac{1}{2}(2A + B + \epsilon), \end{aligned} \tag{15}$$

где $\epsilon = 0$, 1 или 2 в зависимости от условий на нижней и верхней границах.

Предположим, что N представляет, по существу, константу, а A, B, C и ϵ — случайные величины, которые достигают стационарного распределения после освобождения блока и несколько отличающегося стационарного распределения после выделения блока. Среднее изменение величины M при освобождении блока равно среднему значению $(C-A)/N$; среднее изменение величины M при выделении блока равно $1-p$. Таким образом, с учетом достижения состояния равновесия получаем, что $C - A - N + pN = 0$. Однако тогда среднее значение величины $2M$ равно pN плюс среднее значение ϵ, поскольку $2M = N + A - C + \epsilon$ согласно (15). Отсюда следует правило "50%".

Наши предположения о том, что каждое удаление применяется к случайному выделенному блоку, будет справедливо, если время жизни блока представляет собой экспоненциально распределенную случайную величину. С другой стороны, если все блоки имеют примерно одно и то же время жизни, сделанное предположение ложно. Джон Э. Шор (John E. Shore) указал, что блоки типа A обычно "старее" блоков типа C, если характер процесса выделения и освобождения близок к очереди ("первым вошел — первым вышел"; FIFO), поскольку последовательность смежных блоков при этом стремится выстроиться в порядке от "младших" блоков к "старшим" и последний выделенный блок почти никогда не оказывается блоком типа A. Такая

тенденция приводит к наличию меньшего числа доступных блоков, давая даже лучшую производительность по сравнению с производительностью, предсказываемой по правилу "50%" [см. *CACM* **20** (1977), 812–820].

Более детально правило "50%" рассматривается в работах D. J. M. Davies, *BIT 20* (1980), 279–288; C. M. Reeves, *Comp. J.* **26** (1983), 25–35; G. Ch. Pflug, *Comp. J.* **27** (1984), 328–333.

Если не учитывать это интересное правило, наши знания о производительности алгоритмов динамического распределения памяти почти полностью базируются на экспериментах по методу Монте-Карло. При выборе алгоритма выделения памяти для конкретных машины и класса приложения (или конкретного приложения) поучительно провести собственные моделирующие динамическое распределение памяти эксперименты. Автор провел ряд таких экспериментов непосредственно перед написанием этого раздела (и правило "50%" было замечено во время этих экспериментов до того, как было найдено его доказательство). Рассмотрим здесь вкратце методы и результаты проведенных экспериментов.

Основная моделирующая программа работает следующим образом. В начальный момент работы программы, когда значение TIME равно нулю, доступна вся память.

P1. Увеличить TIME на 1.

P2. Освободить все блоки в системе, которые должны быть освобождены при текущем значении TIME.

P3. Вычислить две величины, S (случайный размер) и T (случайное время жизни), основанные на некотором распределении вероятностей, с помощью методов из главы 3.

P4. Выделить новый блок длиной S, который необходимо освободить в момент (TIME + T). Вернуться к шагу P1. ▮

Каждый раз по достижении переменной TIME значения, кратного 200, выводились детальные статистические данные о производительности алгоритмов выделения и освобождения. Для каждой пары алгоритмов использовалась одна и та же последовательность значений S и T. После того как величина TIME превышала 2000, система обычно приходила в более или менее стабильное состояние с неизменными показателями. Однако иногда на шаге P3 алгоритм прекращал свою работу из-за невозможности выделить необходимый объем памяти.

Пусть C — общее количество доступных ячеек памяти и пусть $\bar{S}$ и $\bar{T}$ — средние величины S и T на шаге P3. Легко видеть, что ожидаемое количество занятых слов памяти в каждый момент составляет $\bar{S}\bar{T}$ при достаточно большом значении TIME. Когда в экспериментах $\bar{S}\bar{T}$ было больше, чем $\frac{2}{3}C$, обычно происходило переполнение памяти (часто прежде, чем в действительности происходил запрос на выделение C слов памяти). Память можно было заполнить на 90% при малом по сравнению с C размере блока, но когда размеры блока могли превысить $\frac{1}{3}C$ (вместе с блоками меньших размеров), наблюдалась тенденция программы к "заполнению" памяти при реальном использовании менее $\frac{1}{2}C$ ячеек памяти. Эмпирические результаты свидетельствуют о том, что *для эффективной работы систем динамического распределения памяти не должны использоваться блоки размером свыше* $\frac{1}{10}C$.

Причину такого поведения можно понять, если вспомнить правило "50%": если система достигает состояния равновесия, при котором размер среднего свободного блока f меньше размера среднего используемого блока r, то можно ожидать получения невыполнимого запроса, кроме ситуации, когда "на всякий пожарный случай" имеется достаточно большой свободный блок. Следовательно, в насыщенных системах без переполнения $f \geq r$, и мы имеем $C = fM + rN \geq rM + rN \approx \left(\frac{1}{2}p + 1\right)rN$. Общее количество использованной памяти, таким образом, составляет $rN \leq C/\left(\frac{1}{2}p+1\right)$. Когда $p \approx 1$, использовать больше примерно $\frac{2}{3}$ ячеек памяти невозможно.

Эксперименты по моделированию систем динамического выделения памяти проводились с тремя распределениями размера блоков S:

($S1$) равновероятный выбор целого числа из диапазона от 100 до 2000;

($S2$) выбор размера (1, 2, 4, 8, 16, 32) с вероятностями $(\frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \frac{1}{16}, \frac{1}{32}, \frac{1}{32})$ соответственно;

($S3$) равновероятный выбор размера из множества (10, 12, 14, 16, 18, 20, 30, 40, 50, 60, 70, 80, 90, 100, 150, 200, 250, 500, 1000, 2000, 3000, 4000).

Время T обычно представляло собой случайное целое равномерно распределенное число в диапазоне от 1 до t для фиксированного $t = 10$, 100 или 1000.

Кроме того, были проведены эксперименты, в которых T на шаге P3 представляло собой случайное число, равномерно распределенное в диапазоне от 1 до $\min\left(\lfloor\frac{5}{4}U\rfloor, 12500\right)$, где U — количество единиц времени, оставшегося до ближайшего освобождения занятого блока из имеющихся в настоящий момент в системе. Это распределение времени использовалось для моделирования ситуации "почти последним выделен — первым освобожден" (almost-last-in-first-out): если T всегда выбирается $\leq U$, система выделения памяти выродится в простую стековую операцию, не требующую использования сложных алгоритмов (см. упр. 1). Указанное распределение вынуждает выбранное T быть бóльшим, чем U, около 20% раз, так что мы имеем *почти* стековую операцию. При использовании такого распределения алгоритмы, подобные алгоритмам A, B и C, ведут себя гораздо лучше, чем обычно; очень редко возникало (если возникало вообще) более двух блоков в списке AVAIL, в то время как выделялось около 14 блоков. С другой стороны, алгоритмы системы двойников, R и S, при использовании такого распределения оказывались медленнее, поскольку в стекообразных операциях приходится чаще, чем обычно, расщеплять и сливать блоки. Вывод теоретических свойств этих распределений слишком сложен (см. упр. 32).

На рис. 42 была приведена конфигурация памяти в момент TIME = 5000 с использованием распределения размеров ($S1$) и с равномерно распределенным между 1 и 100 временем при выделении по методу первого подходящего (алгоритмы A и B). В этом эксперименте вероятность p, которая входит в правило "50%", равна 1, так что можно было бы ожидать, что количество свободных блоков составит около половины количества выделенных блоков. На самом деле на рис. 42 показан 21 свободный и 53 выделенных блока. Это не опровергает правило "50%"; например, в момент TIME = 4600 имелось 25 свободных и 49 выделенных блоков. Конфигурация, приведенная на рис. 42, просто показывает, что правило "50%" подвержено статистическим отклонениям. Число свободных блоков, в целом, находится в диа-

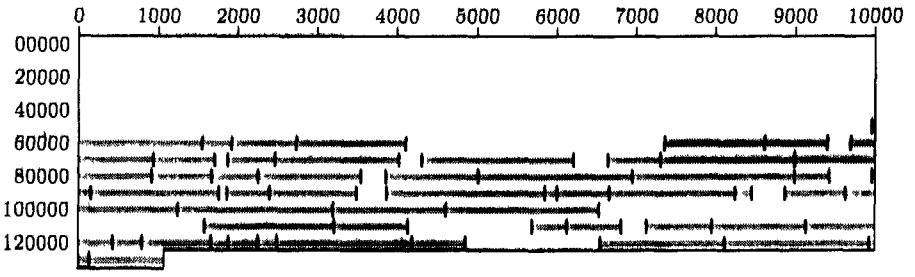


Рис. 43. Карта памяти, полученная с помощью метода наилучшего подходящего. (Сравните ее с картой, представленной на рис. 42, которая показывает результат работы метода первого подходящего, и с рис. 44, на котором в виде дерева представлена карта памяти в результате работы системы двойников с той же последовательностью запросов.)

пазоне от 20 до 30, в то время как количество выделенных блоков находится в диапазоне от 45 до 55.

На рис. 43 приведена конфигурация памяти, полученная *при тех же данных, что и конфигурация на рис. 42*, но с использованием метода наилучшего подходящего. Константа c из шага A4′ была принята равной 16 для устранения малых блоков, и как результат — вероятность p упала до 0.7 и количество свободных областей уменьшилось.

При изменении распределения времени в диапазоне от 1 до 1000 (вместо диапазона от 1 до 100) ситуация была аналогична показанной на рис. 42 и 43, но все соответствующие величины были умножены приблизительно на 10. Например, в ситуации, аналогичной показанной на рис. 42, было 515 выделенных и 240 свободных блоков; в ситуации, подобной приведенной на рис. 43, имелось 176 свободных блоков.

Во всех экспериментах по сравнению методов первого подходящего и наилучшего подходящего последний всегда превосходит первый. После исчерпания размера памяти метод первого подходящего в большинстве случаев остается работоспособным дольше метода наилучшего подходящего.

Система двойников испытывалась с теми данными, которые привели к результатам, изображенным на рис. 42 и 43. Итоговые данные представлены на рис. 44. Здесь все размеры в диапазоне от 257 до 512 рассматривались как 512, в диапазоне от 513 до 1024 — как 1024 и т. д. В среднем это означает увеличение запроса на одну треть (см. упр. 21); система двойников, конечно, лучше работает при распределении размеров (*S2*), а не при распределении (*S1*). Обратите внимание на то, что на рис. 44 имеются доступные блоки размеров 2^9, 2^{10}, 2^{11}, 2^{12}, 2^{13} и 2^{14}.

Моделирование системы двойников показало, что ее производительность выше, чем ожидалось. Ясно, что иногда эта система позволяет иметь два смежных свободных блока одинакового размера без их слияния (если они не являются двойниками). Но такая ситуация не представлена на рис. 44 и в действительности крайне редка на практике. Переполнение памяти происходило при выделении 95% памяти, что отражает удивительно хороший баланс выделения памяти. Кроме того, очень редко требуется разделение блоков в алгоритме R (и их слияние в алгоритме S); дерево остается очень похожим на изображенное на рис. 44 со свободными блоками на чаще всего используемых уровнях. Некоторые математические результаты, позво-

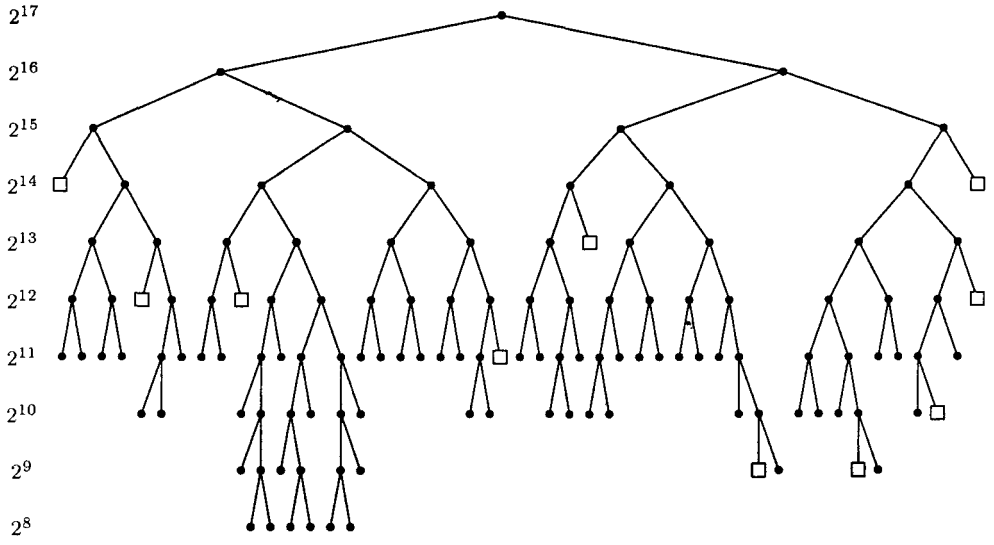


Рис. 44. Карта памяти, полученная при помощи системы двойников. (Структура дерева указывает на деление некоторых больших блоков на двойников половинного размера. Квадратами помечены свободные блоки.)

ляющие понять такое поведение на нижних уровнях дерева, были получены в работе P. W. Purdom, Jr., and S. M. Stigler, *JACM* **17** (1970), 683–697.

Еще одним сюрпризом стало превосходное поведение алгоритма A после модификации, описанной в упр. 6. Потребовалось в среднем только 2.8 проверки размера свободного блока при использовании распределения размеров (*S1*) и времени, равномерно распределенного между 1 и 1000, а более чем в половине случаев требовалось минимальное значение — одна итерация. Все это выполнялось несмотря на наличие около 250 доступных блоков. Тот же эксперимент с немодифицированным алгоритмом A показал, что в среднем необходимо около 125 итераций (т. е. каждый раз проверяется половина списка AVAIL); 200 и более проверок понадобилось примерно в каждом пятом случае.

Такое поведение немодифицированного алгоритма A в действительности может быть предсказано как следствие правила "50%". При равновесии в части памяти, содержащей последнюю половину выделенных блоков, находится также последняя половина свободных блоков; эта часть требует половину времени при освобождении блока и соответственно должна использоваться в половине случаев выделения памяти для поддержки состояния равновесия. Это же обоснование применимо и при замене половины любой другой частью (данные замечания были сделаны Д. М. Робсоном (J. M. Robson)).

Упражнения к этому разделу включают MIX-программы для двух основных методов, которые рекомендуются к использованию на основе приведенных выше замечаний: (i) модифицированный согласно упр. 12 и 16 алгоритм A (система граничных дескрипторов) и (ii) система двойников. Вот приближенные результаты

работы этих методов.

	Время выделения	Время освобождения
Система граничных дескрипторов:	$33 + 7A$	18, 29, 31 или 34
Система двойников:	$19 + 25R$	$27 + 26S$

Здесь $A \geq 1$ — необходимое для поиска достаточно большого блока количество итераций, $R \geq 0$ — количество разбиений блока на два (начальная разность $j - k$ в алгоритме R) и $S \geq 0$ — количество слияний двойников при работе алгоритма S. Моделирующие эксперименты указывают, что при данных предположениях относительно распределения размеров (*S1*) и времени, выбираемом в диапазоне от 1 до 1000, можно получить в среднем $A = 2.8$, $R = S = 0.04$. (Средние значения для распределения времени "почти последним выделен — первым освобожден", которое описано выше, составляют $A = 1.3$, $R = S = 0.9$.) Это показывает, что оба метода весьма быстры (система двойников для MIX несколько быстрее). Напомним, что для системы двойников необходимо примерно на 44% больше памяти, если размеры блоков не ограничены степенями 2.

Соответствующее время для выполнения алгоритма сборки мусора и уплотнения из упр. 33 составляет около 104 единиц при поиске свободного узла, если предположить, что сборка мусора происходит в момент заполненности памяти примерно наполовину, а узлы имеют среднюю длину 5 слов с двумя связями на узел. "За" и "против" этого метода обсуждаются в разделе 2.3.5. Когда память не сильно загружена и выполняются соответствующие ограничения, сборка мусора и уплотнение весьма эффективны; например, на компьютере MIX метод сборки мусора быстрее двух других, если память не заполнена более чем на треть и узлы относительно малы.

Если удовлетворяются условия, лежащие в основе метода сборки мусора, наилучшей стратегией может оказаться разделение пула памяти на две половины и выполнение всех последующих выделений памяти в одной половине. Вместо освобождения становящихся неиспользуемыми блоков мы просто ожидаем, пока текущая половина памяти не заполнится. Затем можно скопировать все активные данные в другую половину, одновременно удаляя все "дыры" между блоками при помощи метода, подобного приведенному в упр. 33. Размер каждой половины пула может изменяться при переключении с одной половины на другую.

Упомянутые выше технологии были применены и к другим алгоритмам выделения памяти. Однако эти методы оказались настолько плохи по сравнению с алгоритмами, описанными в настоящем разделе, что здесь будет дано только их краткое описание.

a) Для каждого размера используется свой список AVAIL. Единый свободный блок при необходимости разбивается на два меньших блока, но никаких попыток вновь объединить такие блоки не предпринимается. Карта памяти становится фрагментированной на все меньшие и меньшие части, пока не принимает совершенно ужасный вид. Простая схема наподобие этой практически эквивалентна раздельному распределению в несвязанных областях, по одной области для каждого размера блока.

b) Была предпринята попытка выполнения двухуровневого выделения. Память делилась на 32 больших сектора. Для выделения больших блоков размером 1, 2

или 3 (очень редко — бóльших размеров) соседних секторов использовался метод выделения памяти "в лоб". Каждый такой большой блок разделялся для удовлетворения запросов на выделение памяти до тех пор, пока в текущем большом блоке не оставалось памяти (при этом начинал использоваться другой большой блок). Каждый большой блок возвращался в свободную память только после освобождения *всех* выделенных из него блоков памяти. Данный метод почти всегда быстро приводит к нехватке памяти.

Хотя этот частный метод двухуровневого выделения и был неработоспособен с использовавшимися автором данными при моделировании динамического распределения, возникают ситуации (на практике очень нечастые), когда многоуровневая стратегия может оказаться предпочтительной. Например, если большая программа работает в несколько стадий, возможно, что на разных стадиях используются узлы разных типов и отдельные типы узлов применяются только в отдельных подпрограммах. В некоторых программах может оказаться желательным использование нескольких различных стратегий распределения памяти для различных классов узлов. Идея выделения памяти по зонам, быть может, с помощью различных стратегий в каждой зоне и с возможностью освобождения всей зоны, рассматривается в работе Douglas T. Ross, *CACM* **10** (1967), 481–492.

Другие эмпирические результаты, полученные при изучении динамического распределения памяти, приводятся в следующих работах: B. Randell, *CACM* **12** (1969), 365–369, 372; P. W. Purdom, S. M. Stigler, and T. O. Cheam, *BIT* **11** (1971), 187–195; B. H. Margolin, R. P. Parmelee, and M. Schatzoff, *IBM Systems J.* **10** (1971), 283–304; J. A. Campbell, *Comp. J.* **14** (1971), 7–9; John E. Shore, *CACM* **18** (1975), 433–440; Norman R. Nielsen, *CACM* **20** (1977), 864–873.

***E. Метод распределенного подходящего.** Если распределение размеров блоков известно заранее и все блоки освобождаются равновероятно, независимо от момента их выделения, можно использовать технологию (которая в данных предположениях превосходит технологии общего назначения), предложенную Э. Г. Коффманом (мл.) (E. G. Coffman, Jr.) и Ф. Т. Лейтоном (F. T. Leighton) [*J. Computer and System Sci.* **38**, (1989), 2–35]. Их "метод распределенного подходящего" (distributed-fit method) работает путем разделения памяти на примерно $N + \sqrt{N}\lg N$ слотов, где N — максимальное число блоков, которые будут обрабатываться в установившемся состоянии. Каждый слот имеет фиксированный размер, хотя различные слоты могут иметь различные размеры. Главное заключается в том, что любой слот имеет фиксированные границы, и он может либо быть пустым, либо содержать единственный выделенный блок.

Первые N слотов схемы Коффмана-Лейтона расположены в соответствии с заданным распределением размеров, а оставшиеся $\sqrt{N}\lg N$ слотов имеют максимальный размер. Например, если предположить, что размеры блоков равномерно распределены между 1 и 256 и если необходимо обработать $N = 2^{14}$ таких блоков, следует разделить память на $N/256 = 2^6$ слотов каждого размера 1, 2, ..., 256, за которыми следует "область переполнения", содержащая $\sqrt{N}\lg N = 2^7 \cdot 14 = 1792$ блоков размером 256. Когда система работает при полной загрузке, ожидается работа с N блоками среднего размера $\frac{257}{2}$, занимающими $\frac{257}{2}N = 2^{21} + 2^{13} = 2\,105\,344$ ячеек памяти (это количество памяти, выделенное для первых N слотов). Кроме

того, имеется $1792 \cdot 256 = 458\,752$ ячеек памяти для обработки случайных отклонений; эти дополнительные накладные расходы составляют $O(N^{-1/2} \log N)$ от общего объема памяти, в отличие от пропорциональных N в случае системы двойников, и становятся незначимыми при $N \to \infty$. В нашем примере, однако, накладные расходы остаются на уровне 18% от общего количества памяти.

Слоты должны быть расположены в таком порядке, чтобы меньшие из них предшествовали бо́льшим. При таком расположении можно выделять блоки с помощью либо технологии первого подходящего, либо технологии наилучшего подходящего (в данном случае оба метода эквивалентны, поскольку размеры слотов упорядочены). При наших предположениях действие описанной методики заключается в начале поиска в, по сути, случайном месте среди первых N слотов при новом запросе на выделение памяти и его продолжении до тех пор, пока не будет найден пустой слот.

Если начальный слот для каждого поиска действительно выбирается случайным образом между 1 и N, частых вторжений в область переполнения не будет. В самом деле, если вставить ровно N элементов, начиная со случайных слотов, переполнение будет встречаться в среднем только $O(\sqrt{N})$ раз. Объяснение этого факта заключается в том, что можно сравнить данный алгоритм с хешированием с линейным исследованием (алгоритм 6.4L), которое имеет то же самое поведение, но поиск пустой ячейки возвращается от N к 1 вместо перехода в область переполнения. Анализ алгоритма 6.4L в теореме 6.4K показывает, что при вставке N элементов среднее смещение каждого элемента от его хеш-адреса составляет $\frac{1}{2}(Q(N) - 1) \sim \sqrt{\pi N/8}$. Из круговой симметрии следует, что это среднее значение — то же, что и среднее количество переходов поиска от слота k к слоту $k+1$ для каждого k. Переполнения в методе распределенного подходящего соответствуют поискам, переходящим от слота N к слоту 1, с той лишь разницей, что наша ситуация даже лучше, поскольку некоторого переполнения можно избежать без возвратов к началу. Таким образом, в среднем встречается менее $\sqrt{\pi N/8}$ переполнений. В этом анализе не приняты во внимание удаления, которые сохраняют предположения алгоритма 6.4L только в случае, когда мы перемещаем блоки назад при удалении другого блока, находящегося между их начальными и выделенными слотами (см. алгоритм 6.4R). Кроме того, однако, их перемещение назад только увеличивает вероятность переполнения. Наш анализ также некорректен для подсчета влияния одновременного наличия более N блоков; это может случиться, если предположить, что время между выделениями блоков составляет $1/N$ от времени их жизни. Если имеется более N блоков, необходим расширенный анализ алгоритма 6.4L, но Коффман и Лейтон доказали, что область переполнения почти никогда не потребует больше чем $\sqrt{N} \lg N$ слотов; вероятность завершения работы составляет менее $O(N^{-M})$ для всех M.

В нашем примере начальный слот для поиска во время выделения не выбирается равномерно среди слотов $1, 2, \ldots, N$. Вместо этого он равномерно выбирается среди слотов $1, 65, 129, \ldots, N-63$, поскольку имеется $N/256 = 64$ слотов каждого размера. Но такое отклонение от рассмотренной в предыдущем разделе случайной модели делает переполнение даже менее вероятным, чем предсказано. Впрочем, не стоит держать пари, если нарушаются предположения о распределении размера блоков и времени их занятости.

F. Переполнение. Что необходимо предпринять, если больше нет свободного пространства? Предположим, что пришел запрос на n последовательных слов, в то время как все блоки слишком малы. В первый раз, когда такое происходит, обычно имеется более чем n доступных ячеек памяти, однако они расположены не последовательно. Уплотнение памяти (т. е. перемещение некоторых используемых ячеек, чтобы доступные ячейки памяти были собраны вместе) могло бы позволить продолжать обработку запросов. Однако уплотнение — медленный процесс, который требует строгой дисциплины применения указателей*. Кроме того, в подавляющем большинстве случаев при использовании метода первого подходящего независимо от того, сколько раз проводилось уплотнение, память в конечном счете оказывается полностью исчерпанной. Таким образом, вообще говоря, не имеет смысла писать программы для уплотнения, за исключением специальных случаев, связанных со сборкой мусора (см. упр. 33). Если переполнение ожидается заранее, можно "подстелить соломки", воспользовавшись некоторыми из методов переноса элементов из оперативной памяти на внешние запоминающие устройства с обеспечением возврата элемента в оперативную память при необходимости в нем**. Отсюда вытекает, что для эффективной работы все программы, работающие с областями динамической памяти, должны быть строго ограничены в плане допустимых ссылок на другие блоки, а также обеспечиваться аппаратно (например, должна иметься возможность генерирования прерывания при отсутствии данных в оперативной памяти или автоматической подкачке страниц).

Необходима некоторая процедура принятия решения о том, какие блоки являются наиболее подходящими кандидатами на удаление из оперативной памяти. Одна из идей состоит в содержании двусвязного списка выделенных блоков, при этом каждый блок двигается к началу списка при обращении к нему. Таким образом, блоки оказываются рассортированными в порядке последнего к ним обращения и блоки, расположенные в конце списка, удаляются первыми. Подобного эффекта можно достичь более просто: необходимо поместить выделенный блок в циклический список и включить в каждый блок бит "недавно использован". Этот бит устанавливается равным 1 при обращении к блоку. Когда наступает время удаления блока, указатель перемещается по циклическому списку, сбрасывая все биты "недавно использован" в нуль, пока не будет найден блок, к которому не было обращений со времени последнего прохода указателя по этой части списка.

Д. М. Робсон (J. M. Robson) показал [*JACM* **18** (1971), 416–423], что стратегии динамического выделения памяти, которые никогда не переносят выделенные блоки, не могут гарантировать эффективное использование памяти. Всегда найдутся патологические ситуации, при которых метод перестанет работать. Например, даже когда блоки ограничены размерами 1 или 2, переполнение может произойти при заполнении памяти примерно на 2/3, какой бы алгоритм не использовался! Интересные результаты Робсона рассматриваются в упр. 36–40, а также в упр. 42

---

* В этом случае справедливы все замечания, сделанные выше в связи с методом сборки мусора. — *Прим. перев.*

** Этот процесс известен в современной литературе как *свопинг* (*swapping*). Еще одно замечание заключается в том, что внешним запоминающим устройством, в принципе, может быть и сама оперативная память — стоит только вспомнить, каким образом в DOS использовалась недоступная для прямой адресации память EMS/XMS. — *Прим. перев.*

и 43, в которых показано, что метод наилучшего подходящего имеет очень плохой наихудший случай по сравнению с методом первого подходящего.

G. Для дальнейшего чтения. Всестороннее исследование и критический обзор технологий динамического выделения памяти, основанный на более многолетнем опыте, чем опыт автора этой книги, можно найти в работе Paul R. Wilson, Mark S. Johnstone, Michael Neely, and David Boles, *Lecture Notes in Computer Science* **986** (1995), 1–116.

УПРАЖНЕНИЯ

1. [*20*] Какие упрощения алгоритмов выделения и освобождения памяти, описанных в этом разделе, можно сделать, если предположить, что запросы на выделение и освобождение памяти всегда приходят в стековом порядке ("последним выделен — первым освобожден"), т. е. ни один блок не освобождается, пока не освобождены все выделенные после него блоки?

2. [*HM23*] (Э. Вольман (E. Wolman).) Предположим, что необходимо выбрать фиксированный размер узла для элементов переменной длины, и предположим также, что, когда каждый узел имеет длину k, а элемент — длину l, для хранения такого элемента используется $\lceil l/(k-b) \rceil$ узлов (здесь b — константа, обозначающая, что b слов каждого узла содержат управляющую информацию, например связь со следующим узлом). Если средняя длина l элемента равна L, то какой выбор k минимизирует среднее количество необходимой памяти? (Положим, что среднее значение $(l/(k-b)) \bmod 1$ равно $1/2$ для любого фиксированного k и переменного l.)

3. [*40*] При помощи компьютерного моделирования сравните следующие методы выделения памяти: наилучшего подходящего, первого подходящего и *наихудшего подходящего*. В последнем случае всегда выбирается наибольший доступный блок. Есть ли при этом существенная разница в использовании памяти?

4. [*22*] Напишите MIX-программу для алгоритма A, обращая особое внимание на ускорение работы внутреннего цикла. Положите, что поле SIZE — это (4:5), поле LINK — (0:2), а $\Lambda < 0$.

▶ **5.** [*18*] Предположим, известно, что в алгоритме A N всегда не меньше 100. Стóит ли устанавливать $c = 100$ на модифицированном шаге A4′?

▶ **6.** [*23*] (*Следующий подходящий.*) При постоянном использовании алгоритма A возникает тенденция к тому, что блоки малого размера остаются в начале списка AVAIL, поэтому приходится часто проводить длительный поиск блока нужного размера. Например, обратите внимание на рис. 42, на котором четко видно, как увеличиваются размеры блоков (как занятых, так и свободных) от начала памяти к ее концу. (Список AVAIL рассортирован по увеличению адресов памяти, как того требует алгоритм B.) Можете ли вы предложить такой вариант модификации алгоритма A, что (a) короткие блоки при его работе не скапливаются в некоторой области и (b) список AVAIL остается упорядоченным по адресам памяти для работы алгоритма наподобие B?

7. [*10*] Пример (1) показывает, что иногда метод первого подходящего заведомо превосходит метод наилучшего подходящего. Приведите аналогичный пример, когда метод наилучшего подходящего заведомо превосходит метод первого подходящего

8. [*21*] Покажите, каким образом можно просто модифицировать алгоритм A для работы по методу наилучшего подходящего (вместо изначального метода первого подходящего).

▶ **9.** [*26*] Каким образом следует разработать алгоритм выделения памяти, работающий по методу наилучшего подходящего, чтобы он не проходил в поисках необходимого блока

памяти весь список AVAIL? (Попытайтесь придумать метод, снижающий необходимый поиск настолько, насколько это возможно.)

10. [*22*] Покажите, каким образом можно модифицировать алгоритм B, чтобы блок из N последовательных ячеек, начинающийся с адреса P0, становился свободным без предположения о занятости всех N ячеек. Считается, что освобождаемая область может перекрывать несколько уже освобожденных блоков.

11. [*M25*] Покажите, что предложенное в упр. 6 усовершенствование алгоритма A можно также использовать для некоторого улучшения алгоритма B, что приведет к снижению средней длины поиска от половины длины списка AVAIL до одной трети. (Предполагается, что освобождающийся блок будет вставлен в упорядоченный список AVAIL в случайном месте.)

▶ **12.** [*20*] Модифицируйте алгоритм A таким образом, чтобы он следовал соглашениям "помеченных границ" (7)–(9), использовал измененный шаг A4′, описанный в тексте раздела, и включал усовершенствования из упр. 6.

13. [*21*] Напишите MIX-программу для алгоритма из упр. 12.

14. [*21*] Какие отличия появились бы в алгоритме C и алгоритме из упр. 12, если бы (a) в последнем слове свободного блока отсутствовало поле SIZE и (b) поле SIZE отсутствовало в первом слове выделенного блока?

▶ **15.** [*24*] Покажите, как ускорить работу алгоритма C за счет небольшого удлинения программы, не изменяя связей больше, чем это абсолютно необходимо в каждом из четырех случаев, в зависимости от того, чем является каждый из дескрипторов TAG(P0 − 1) и TAG(P0 + SIZE(P0)) — плюсом или минусом.

16. [*24*] Напишите MIX-программу для алгоритма C, включающую идеи из упр. 15.

17. [*10*] Каким должно быть содержимое LOC(AVAIL) и LOC(AVAIL)+1 в (9) при отсутствии доступных блоков?

▶ **18.** [*20*] Рис. 42 и 43 получены с помощью одних и тех же данных и по сути одинаковых алгоритмов (алгоритмов A и B), но рис. 43 подготовлен модифицированным алгоритмом A с выбором наилучшего подходящего вместо первого подходящего. Почему при этом на рис. 42 большая свободная область находится в *старших* адресах памяти, в то время как на рис. 43 она же находится в *младших* адресах?

▶ **19.** [*24*] Предположим, что блоки памяти имеют вид (7), но без полей TAG или SIZE в последнем слове блока. Предположим далее, что для освобождения блока используется следующий алгоритм: Q ← AVAIL, LINK(P0) ← Q, LINK(P0 + 1) ← LOC(AVAIL), LINK(Q + 1) ← P0, AVAIL ← P0, TAG(P0) ← "−". (Он не выполняет объединение соседних свободных областей.)

Разработайте алгоритм для выделения памяти, аналогичный алгоритму A, который выполняет объединение смежных свободных блоков во время поиска в списке AVAIL и при этом исключает любую излишнюю фрагментацию памяти, как в (2)–(4).

20. [*00*] Почему в системе двойников вместо линейных списков желательно иметь двусвязные списки AVAIL[*k*]?

21. [*HM25*] Исследуйте отношение a_n/b_n при $n \to \infty$, где a_n — сумма первых n членов ряда $1 + 2 + 4 + 4 + 8 + 8 + 8 + 8 + 16 + 16 + \cdots$, а b_n — сумма первых n членов ряда $1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 10 + \cdots$.

▶ **22.** [*21*] В тексте раздела неоднократно упоминалось, что система двойников позволяет иметь только блоки размеров 2^k, и упр. 21 показывает, что это может привести к существенному увеличению требуемой памяти. Но если приходит запрос на блок размером 11 слов, то почему нельзя найти блок размером 16 слов и разделить его на выделяемый блок размером 11 слов и два свободных блока с размерами 4 и 1 слово?

23. [*05*] Каков двоичный адрес двойника блока размером 4, двоичный адрес которого равен 011011110000? Каким бы он был, если бы блок имел размер 16?

24. [*20*] Согласно приведенному в тексте раздела алгоритму наибольший блок (размером 2^m) не имеет двойника, так как представляет собой всю память. Будет ли правильным определение

$$\text{двойник}_m(0) = 0$$

(по сути, блок станет собственным двойником), и можно ли таким образом избежать проверки $k = m$ на шаге S1?

▸ **25.** [*22*] Раскритикуйте следующую идею: "Динамическое выделение памяти с помощью системы двойников на практике никогда не приводит к выделению блока размером 2^m (поскольку этим исчерпывается вся память), и, вообще говоря, имеется некоторый максимальный размер 2^n, такой, что блоки большего размера никогда не выделяются. Значит, начинать работу с такого размера блоков — пустая трата времени, как и объединение в алгоритме S блоков, образующих в результате свободный блок размером, превосходящим 2^n".

▸ **26.** [*21*] Поясните, каким образом можно использовать систему двойников для динамического выделения памяти с адресами от 0 до $\texttt{M} - 1$ даже в случае, когда M не имеет вид 2^m, как требуется в тексте раздела.

27. [*24*] Напишите MIX-программу для алгоритма R и определите время ее работы.

28. [*25*] Напишите MIX-программу для алгоритма S и определите время ее работы, считая MIX двоичным компьютером с новой операцией XOR, с помощью обозначений из раздела 1.3.1 определяемой так: "C = 5, F = 5. Для каждого бита в ячейке M, равного 1, соответствующий бит в регистре A дополняется (изменяется с 0 на 1 или с 1 на 0); знак rA остается неизменным, время выполнения равно $2u$"

29. [*20*] Может ли система двойников работать без бита дескриптора в каждом выделенном блоке?

30. [*M48*] Проанализируйте среднее поведение алгоритмов R и S, задавая обоснованные распределения для последовательности запросов на выделение памяти.

31. [*M40*] Можно ли построить аналогичную системе двойников систему динамического распределения памяти, основанную на последовательности чисел Фибоначчи, а не на степенях двойки? (Таким образом, можно начать с F_m свободных слов и разделить доступные блоки из F_k слов на два двойника длиной F_{k-1} и F_{k-2} соответственно.)

32. [*HM47*] Определите $\lim_{n\to\infty} \alpha_n$, если он существует, где α_n — среднее значение t_n в случайной последовательности, определенной таким образом. Даны значения t_k для $1 \le k < n$; t_n равновероятно выбирается из множества значений $\{1, 2, \ldots, g_n\}$, где

$$g_n = \lfloor \tfrac{5}{4} \min(10000,\ f(t_{n-1} - 1),\ f(t_{n-2} - 2),\ \ldots,\ f(t_1 - (n-1))) \rfloor$$

и $f(x) = x$ при $x > 0$ и $f(x) = \infty$ при $x \le 0$. [*Примечание.* Некоторые ограниченные эмпирические тесты показывают, что α_n может быть примерно равно 14, но, вероятно, это не слишком точное значение.]

▸ **33.** [*28*] (*Сборка мусора и уплотнение.*) Положим, что ячейки памяти 1, 2, ..., AVAIL − 1 используются в качестве пула памяти для узлов переменного размера, имеющих следующий вид: первое слово NODE(P) содержит поля

SIZE(P) = количество слов в NODE(P);
T(P) = количество полей связей в NODE(P); T(P) < SIZE(P);
LINK(P) = специальное поле связи для использованиятолько при сборке мусора.

В памяти за NODE(P) следует NODE(P + SIZE(P)). Предположим, что в NODE(P) в качестве связей с другими узлами используются LINK(P + 1), LINK(P + 2), ..., LINK(P + T(P)) и

каждое из этих полей связей представляет собой либо Λ, либо адрес первого слова другого узла. И наконец, предположим, что в программе имеется еще одна переменная связи с именем USE, которая указывает на один из узлов.

Разработайте алгоритм, который (i) определяет все узлы, прямо или косвенно доступные из переменной USE, (ii) перемещает эти узлы в ячейки памяти от 1 до K − 1 для некоторого K, изменяя все связи так, чтобы сохранились структурные отношения, и (iii) устанавливает AVAIL ← K.

Например, рассмотрим следующее содержимое памяти, где INFO(L) обозначает содержимое ячейки L, исключая LINK(L):

1: SIZE = 2, T = 1	6: SIZE = 2, T = 0	AVAIL = 11,
2: LINK = 6, INFO = A	7: CONTENTS = D	USE = 3.
3: SIZE = 3, T = 1	8: SIZE = 3, T = 2	
4: LINK = 8, INFO = B	9: LINK = 8, INFO = E	
5: CONTENTS = C	10: LINK = 3, INFO = F	

Ваш алгоритм должен преобразовать эти данные в

1: SIZE = 3, T = 1	4: SIZE = 3, T = 2	AVAIL = 7,
2: LINK = 4, INFO = B	5: LINK = 4, INFO = E	USE = 1.
3: CONTENTS = C	6: LINK = 1, INFO = F	

34. [*29*] Напишите MIX-программу для алгоритма из упр. 33 и определите время ее работы.

35. [*22*] Сопоставьте методы динамического распределения памяти из этого раздела с технологиями последовательных списков переменного размера, рассмотренными в конце раздела 2.2.2.

▶ **36.** [*20*] В некоторой закусочной в Голливуде (Калифорния) имеется 23 места для посетителей, которые приходят поодиночке или вдвоем. Хозяйка закусочной показывает посетителям их места Докажите, что она всегда сможет рассадить посетителей, не разбивая пары, если одновременно в закусочной будет находиться не более 16 посетителей, а одиночки не сидят на местах 2, 5, 8, ..., 20 (предполагается, что каждая пришедшая пара уходит вместе).

▶ **37.** [*26*] Продолжая упр. 36, докажите, что хозяйка не всегда может так удачно рассадить посетителей, если в закусочной только 22 места. Независимо от используемой ею стратегии может возникнуть ситуация, когда в закусочной будет находиться всего 14 посетителей, но для вошедшей пары не найдется двух соседних пустых мест.

38. [*M21*] (Д. М. Робсон (J. M. Robson).) Задача о закусочной, изложенная в упр. 36 и 37, может быть обобщена для определения производительности в наихудшем случае *для любого алгоритма динамического выделения памяти, который никогда не перемещает* выделенные блоки. Пусть $N(n, m)$ — наименьшее количество памяти, такое, что любая серия запросов на выделение и освобождение может быть выполнена без переполнения в случае, когда все размеры блоков не превышают m, а общее количество затребованной памяти не превосходит n. В упр. 36 и 37 доказано, что $N(16, 2) = 23$. Определите точное значение $N(n, 2)$ для всех n.

39. [*HM23*] (Д. М. Робсон.) Используя обозначения из упр. 38, покажите, что

$$N(n_1 + n_2,\, m) \le N(n_1,\, m) + N(n_2,\, m) + N(2m - 2,\, m).$$

Следовательно, для фиксированного m существует

$$\lim_{n\to\infty} N(n, m)/n = N(m).$$

40. [*HM50*] Продолжая упр 39, определите $N(3)$, $N(4)$ и $\lim_{m\to\infty} N(m)/\lg m$, если таковой существует.

41. [*M27*] Назначение этого упражнения состоит в рассмотрении наихудшего случая использования памяти в системе двойников В частности, плохой случай возникает, например, если начать с пустой памяти и действовать следующим образом сначала выделить $n = 2^{r+1}$ блоков длиной 1, которые будут занимать адреса с 0 по $n-1$, затем для $k = 1, 2, \ldots, r$ освободить все блоки начинающиеся с адресов, которые не делятся на 2^k, и выделить $2^{-k-1}n$ блоков длиной 2^k, которые будут занимать адреса с $\frac{1}{2}(1+k)n$ по $\frac{1}{2}(2+k)n-1$ Для этой процедуры необходимо в $1+\frac{1}{2}r$ раз больше памяти, чем выделено

Докажите, что наихудший случай не может быть существенно хуже этого когда все запросы приходят на блоки размером $1, 2, \ldots, 2^r$ и общий размер запрошенной памяти в любой момент не превышает n, где n кратно 2^r, система двойников никогда не переполнит область памяти размером $(r+1)n$

42. [*M40*] (Д М Робсон, 1975) Пусть $N_{\mathrm{BF}}(n,m)$ — количество памяти, необходимой, чтобы гарантировать отсутствие переполнения при использовании для выделения метода наилучшего подходящего, как в упр 38 Найдите "атакующую" стратегию, показывающую, что $N_{\mathrm{BF}}(n,m) \geq mn - O(n+m^2)$

43. [*HM35*] Продолжая упр 42, положим, что $N_{\mathrm{FF}}(n,m)$ — необходимая при методе первого подходящего память Найдите "оборонительную" стратегию, показывающую, что $N_{\mathrm{FF}}(n,m) \leq H_m n/\ln 2$ (Следовательно, наихудший случай стратегии первого подходящего не так далек от наилучшего из наихудших случаев)

44. [*M21*] Предположим, что функция распределения $F(x)$ = (вероятность того, что размер блока $\leq x$) непрерывна Например, $F(x)$ равна $(x-a)/(b-a)$ при $a \leq x \leq b$, если размеры равномерно распределены между a и b Приведите формулу, выражающую размеры первых N слотов при использовании метода распределенного подходящего

2.6. ИСТОРИЯ И БИБЛИОГРАФИЯ

Линейные списки и прямоугольные массивы информации, содержащиеся в последовательных ячейках памяти, широко использовались с первых дней появления компьютеров с хранимой программой, и в самых ранних работах по программированию приведены базовые алгоритмы для прохождения этих структур. [См., например, J. von Neumann, *Collected Works* **5**, 113–116 (написана в 1946 году); M. V. Wilkes, D. J. Wheeler, S. Gill, *The Preparation of Programs for an Electronic Digital Computer* (Reading, Mass.: Addison-Wesley, 1951), subroutine V-1. Обратите особое внимание на работу Конрада Зусе (Konrad Zuse) *Berichte der Gesellschaft für Mathematik und Datenverarbeitung* **63** (Bonn, 1972), написанную в 1945 году. Зусе первым создал нетривиальные алгоритмы для работы со списками динамически изменяемой длины.] До появления индексных регистров работа с последовательными линейными списками выполнялась при помощи арифметических операций над самими машинными командами, и использование такой арифметики стало одним из самых ранних обоснований создания компьютеров, в которых программа разделяла с обрабатываемыми данными общее пространство памяти.

Технологии, позволяющие линейным спискам переменной длины занимать последовательные адреса памяти таким образом, чтобы при необходимости их можно было сдвигать в ту или иную сторону, как описано в разделе 2.2.2, были, повидимому, гораздо более поздним открытием. Д. Данлэп (J. Dunlap) из Digitek Corporation разработал такие технологии до 1963 года в связи с созданием ряда компилирующих программ. Примерно в то же время данная идея независимо возникла при создании компилятора COBOL в IBM Corporation и набора связанных с ним подпрограмм, называвшегося CITRUS, который впоследствии использовался в различных компьютерах. Технологии оставались неопубликованными до тех пор, пока не были независимо разработаны Яном Гарвиком (Jan Garwick) из Норвегии; см. *BIT* **4** (1964), 137–140.

Идея размещения линейного списка *не* в последовательных ячейках памяти, видимо, появилась в связи с разработкой компьютеров с устройствами хранения информации на магнитных барабанах. После выполнения команды из ячейки n такой компьютер обычно не был готов к работе с командой из следующей ячейки $n + 1$, так как барабан уже прошел эту точку. В зависимости от выполняющейся команды наиболее благоприятным местом расположения следующей команды может оказаться, например, ячейка $n + 7$ или $n + 18$, и при оптимальном размещении команд быстродействие машины может увеличиться в 6–7 раз. [Обсуждение интересных задач, относящихся к оптимальному расположению команд, можно найти в статье автора этой книги в *JACM* **8** (1961), 119–150.] Поэтому в каждой машинной команде появлялось дополнительное адресное поле, служившее связью со следующей командой. Такая идея, названная "адресация 1+1", обсуждалась в работе John Mauchly, *Theory and Techniques for the Design of Electronic Computers* **4** (U. of Pennsylvania, 1946), Lecture 37. В ней в зачаточной форме содержится понятие связанного списка, хотя так часто использовавшиеся нами в этой главе операции динамической вставки и удаления оставались неизвестными. Другое раннее упоминание о связях в программах приведено в меморандуме Г. П. Лана (H. P. Luhn) (1953 г.), в котором для внешнего поиска предлагалось применять "цепочки" (см. раздел 6.4).

Реально технологии использования связанной памяти родились, когда А. Ньювелл (A. Newell), Д. К. Шоу (J. C. Shaw) и Г. А. Саймон (H. A. Simon) начали свои исследования эвристического решения задач с помощью компьютера. Весной 1956 года для написания программ поиска доказательств в математической логике они разработали первый язык обработки списков — IPL-II. (IPL означает Information Processing Language — язык обработки информации.) Это была система, использующая указатели и включающая важные концепции наподобие списка свободного пространства, однако концепция стека тогда еще не была достаточно разработана. Созданный годом позже IPL-III включал команды для помещения в стек и выборки из стека в качестве важных основных операций. [Подробнее о IPL-II речь идет в *IRE Transactions* **IT-2** (September, 1956), 61–70; *Proc. Western Joint Comp. Conf.* **9** (1957), 218–240. Материалы по IPL-III впервые появились в виде конспекта лекций в Университете штата Мичиган летом 1957 года.]

Работа Ньювелла, Шоу и Саймона привлекла многих исследователей к использованию связанной памяти, которую в то время часто называли по первым буквам фамилий авторов NSS-памятью, но, в основном, для решения задач, связанных с моделированием человеческого мышления. Постепенно эти технологии стали базовыми инструментами программирования. Первая статья, описывающая эффективность связанной памяти для выполнения "приземленных" задач, была опубликована Д. В. Карром III (J. W. Carr III) в *CACM* **2**, 2 (February, 1959), 4–6. В ней Карр указал, что связанными списками легко манипулировать на обычных языках программирования без привлечения изощренных подпрограмм или интерпретирующих систем. [См. также G. A. Blaauw, "Indexing and control-word techniques", *IBM J. Res. and Dev.* **3** (1959), 288–301.]

Сначала для связанных таблиц использовались узлы из одного слова, но около 1959 года постепенно различные коллективы убедились в преимуществах узлов из нескольких последовательных слов и "многосвязных" списков. Первой статьей, посвященной сугубо этой идее, была статья D T. Ross, *CACM* **4** (1961), 147–150. В то время для того, что в настоящей главе мы называем узлом, автором указанной статьи использовался термин "плекс" (plex), хотя позднее он употреблял это слово в другом смысле — для описания класса узлов в комбинации с алгоритмами их прохождения.

Обозначения для ссылки на поле в узле, в основном, бывают двух видов: имя поля либо предшествует обозначению указателя, либо следует за ним. Таким образом, то, что в данной главе записывается как "`INFO(P)`", некоторые авторы записывают как "`P.INFO`". Во время создания этой главы обе записи, пожалуй, встречались одинаково часто. Запись, принятая в настоящей книге, имеет то достоинство, что она непосредственно транслируется на языки FORTRAN, COBOL и подобные, если определить массивы `INFO` и `LINK` и использовать в качестве индекса `P`. Кроме того, представляется естественным использовать обозначения математической функции для описания атрибутов узла. Заметьте, что "`INFO(P)`" произносится как "`INFO` от `P`", как принято в математике — $f(x)$ вербально передается как "f от x". Альтернативное обозначение "`P.INFO`" менее естественно, поскольку имеет тенденцию к подчеркиванию `P`, хотя и может быть прочитано как "`P`-е `INFO`". Причина, по которой `INFO(P)` кажется более предпочтительным способом записи, по-видимому, заключается в том, что `P` - переменная, а `INFO` имеет конкрет-

ный смысл при использовании записи. По аналогии можно рассматривать вектор $A = (A[1], A[2], \ldots, A[100])$ как узел, имеющий 100 полей с именами $1, 2, \ldots, 100$. Теперь на второе поле в наших обозначениях можно сослаться как "2(P)", где P указывает на вектор A; но если ссылаться на j-й элемент вектора, то более естественно записать "$A[j]$", помещая переменную "j" второй. Точно так представляется более подходящей запись INFO(P) с переменной "P" на втором месте*.

Возможно, первыми, кто увидели в концепциях стека ("последним вошел — первым вышел"; last-in-first-out) и очереди ("первым вошел — первым вышел"; first-in-first-out) важные объекты изучения, были бухгалтеры, заинтересованные в упрощении процесса начисления подоходных налогов. Обсуждение методов LIFO и FIFO при составлении прайс-листов можно найти в любом учебнике среднего уровня для бухгалтеров [см., например, C. F. and W. J. Schlatter, *Cost Accounting* (New York: Wiley, 1957), Chapter 7]. В середине 40-х годов А. М. Тьюринг (A. M. Turing) разработал механизм стека, названный реверсивной памятью, для связывания подпрограмм, локальных переменных и параметров. Тьюринг использовал для принятых ныне понятий "добавление в стек" и "снятие со стека" (push/pop) понятия "закапывать" (bury) и "откапывать" (disinter/unbury) (см. ссылки в разделе 1.4.5). Несомненно, простые применения стеков, расположенных в последовательных адресах памяти, в программировании с первых дней были обычным делом, поскольку стек представляет собой интуитивно понятную и естественную концепцию. Программирование стеков в связанном виде появилось впервые в IPL, как упоминалось выше; имя "стек" происходит из терминологии IPL (хотя в IPL имелся более официальный термин "pushdown list") и независимо введено Э. В. Дейкстрой (E. W. Dijkstra) [*Numer. Math.* **2** (1960), 312–318]. Термин "дек" (deque) был введен Э. Ю. Швеппе (E. J. Schweppe) в 1966 году.

Происхождение циклических списков и списков с двойными связями не ясно. Вероятно, эти идеи многим показались естественными. Важным фактором в популяризации этих технологий было существование основанных на них систем общего назначения для обработки списков [см. Knotted List Structures, *CACM* **5** (1962), 161–165, и Symmetric List Processor, *CACM* **6** (1963), 524–544, Й. Вейзенбаума (J. Weizenbaum)]. Айвен Сэтерленд (Ivan Sutherland) ввел в использование независимые двусвязные списки в бóльших узлах в своей системе Sketchpad (Ph. D. thesis, Mass. Inst. of Technology, 1963).

С первых "компьютерных" дней многие квалифицированные программисты независимо разработали различные методы адресации и прохождения многомерных массивов информации. Таким образом была рождена еще одна часть неопубликованного компьютерного фольклора. Обзор этой темы можно найти в работе H. Hellerman, *CACM* **5** (1962), 205–207 [см. также J. C. Gower, *Comp. J.* **4** (1962), 280–286].

Структуры деревьев, явно представленные в памяти компьютера, изначально использовались для приложений, работающих с алгебраическими формулами. Ма-

* Аргументация автора безупречна, но... "меняется все в наш век перемен" и место FORTRAN и COBOL заняли языки C++, Pascal и Java (а также Visual Basic и многие другие), в которых полна смысла именно запись P.INFO — "поле INFO структуры P" (или "записи" и т. п. — название варьируется в зависимости от языка программирования). Что же касается аналогии с массивом, то в том же C или C++ на элементы массива a[] можно ссылаться и как на a[j], и как на j[a]. — *Прим. перев.*

шинный язык для ряда ранних компьютеров использовал трехадресный код для представления вычислений арифметических выражений; последний эквивалентен INFO, LLINK и RLINK бинарного представления дерева. В 1952 году Г. Д. Кахриманиан (H. G. Kahrimanian) разработал алгоритмы для дифференцирования алгебраических формул, представленных в расширенном трехадресном коде [см. *Symposium on Automatic Programming* (Washington, D.C.: Office of Naval Research, May, 1954), 6–14].

С тех пор древовидные структуры различных видов независимо изучались многими исследователями в связи с их применением во многих компьютерных приложениях. Однако в публикациях основные технологии для работы с деревьями (не со списками) появлялись лишь в составе описаний отдельных алгоритмов. Первый общий обзор был сделан в связи с изучением структур данных в работе K. E. Iverson and L. R. Johnson [IBM Corp. research reports RC-390, RC-603, 1961]; см. также Iverson, *A Programming Language* (New York: Wiley, 1962), Chapter 3, и G. Salton, *CACM* **5** (1962), 103–114.

Концепция *прошитых* (*threaded*) деревьев принадлежит А. Д. Перлису (A. J. Perlis) и Ч. Торнтону (C. Thornton) [*CACM* **3** (1960), 195–204]. В их статье вводится также важная идея прохода деревьев в различных порядках и приводятся многочисленные примеры алгоритмов алгебраических манипуляций. К сожалению, эта важная статья готовилась наскоро и содержит много опечаток. Прошитые списки Перлиса и Торнтона в нашей терминологии представляют собой правопрошитые деревья. Бинарные деревья, прошитые в *обоих* направлениях, были независимо открыты А. В. Хольтом (A. W. Holt), *A Mathematical and Applied Investigation of Tree Structures* (Thesis, U. of Pennsylvania, 1963). Прямой и непрямой порядки узлов деревьев в работе Z. Pawlak, *Colloquium on the Foundation of Mathematics*, Tihany, 1962 (Budapest: Akadémiai Kiadó, 1965), 227–238, назывались "нормальный прямой порядок" и "дуальный прямой порядок". В приведенной выше работе Айверсон и Джонсон назвали прямой порядок порядком поддеревьев. Графические способы представления связей между древовидными структурами и соответствующими линейными обозначениями были описаны в A. G. Oettinger, *Proc. Harvard Symp. on Digital Computers and their Applications* (April, 1961), 203–224. Представление деревьев в прямом порядке степенями с алгоритмами, связывающими это представление с десятичной записью Дьюи и другими свойствами деревьев, были представлены в работе S. Gorn, *Proc. Symp. Math. Theory of Automata* (Brooklyn: Poly. Inst., 1962), 223–240.

История древовидных структур как математических объектов вместе с библиографией по этой теме приведена в разделе 2.3.4.6.

В то время, когда в 1966 году создавался этот раздел, большинство сведений об информационных структурах было получено благодаря изучению систем обработки списков, игравших очень важную роль в истории. Первой широко используемой системой был язык IPL-V (наследник IPL-III, разработанный в 1959 году). Он представлял собой интерпретирующую систему, в которой программист изучал машиноподобный язык для работы со списками. Примерно в то же время Г. Гелернтером (H. Gelernter) и его сотрудниками был разработан FLPL (набор подпрограмм на языке FORTRAN для работы со списками, который также обязан своему появлению на свет IPL, но в отличие от него вместо интерпретируемого языка использовал вызовы

подпрограмм). Третья система, LISP, была также создана в 1959 году Дж. Мак-Карти (J. McCarthy). LISP существенно отличался от своих предшественников: его программы были (и остаются) математическими функциональными выражениями, скомбинированными с "условными выражениями" (conditional expressions) и конвертированными затем в представление списков. В 60-е годы возникло много систем обработки списков; с исторической точки зрения среди них наиболее известен набор подпрограмм SLIP для реализации списков с двойными связями на языке FORTRAN, созданный Й. Вейзенбаумом (J. Weizenbaum).

В статье Боброу (Bobrow) и Рафаэля (Raphael), *CACM* **7** (1964), 231–240, можно прочесть краткое введение в IPL-V, LISP и SLIP. В ней же дается сравнение этих систем. Превосходное введение в LISP было опубликовано Ф. М. Вудвордом (P. M. Woodward) и Д. П. Дженкинсом (D. P. Jenkins), *Comp. J.* **4** (1961), 47–53. Кроме того, можно ознакомиться с авторским описанием их собственных систем (эти статьи представляют значительную историческую ценность): "An introduction to IPL-V" А. Ньювелла (A. Newell) и Ф. М. Тонге (F. M. Tonge), *CACM* **3** (1960), 205–211; "A FORTRAN-compiled List Processing Language" Г. Гелернтера (H. Gelernter), Д. Р. Хансена (J. R. Hansen) и К. Л. Гербериха (C. L. Gerberich), *JACM* **7** (1960), 87–101; "Recursive functions of symbolic expressions and their computation by machine, I" Джона Мак-Карти (John McCarthy), *CACM* **3** (1960), 184–195; "Symmetric List Processor" Й. Вейзенбаума (J. Weizenbaum), *CACM* **6** (1963), 524–544. Статья Вейзенбаума включает полное описание всех использованных в SLIP алгоритмов. Из всех перечисленных ранних систем только LISP имел все необходимые составляющие для того, чтобы пережить десятилетия дальнейшего прогресса. Мак-Карти описал раннюю историю LISP в *History of Programming Languages* (Academic Press, 1981), 173–197.

В 60-е годы появился ряд систем для *работы со строками*, в которых первостепенное значение приобрели операции со строками переменной длины, содержащими алфавитную информацию — поиск вхождения в строку определенной подстроки, ее замещение другой и т. п. Наиболее важными из них с исторической точки зрения были COMIT [V. H. Yngve, *CACM* **6** (1963), 83–84] и SNOBOL [D. J. Farber, R. E. Griswold, and I. P. Polonsky, *JACM* **11** (1964), 21–30]. Хотя системы работы со строками широко использовались и состоят, в первую очередь, из алгоритмов, подобных рассмотренным в этой главе, они сыграли сравнительно малую роль в истории развития технологий представления информационных структур. Детали внутренней реализации были скрыты от пользователей этих систем. Обзор ранних систем работы со строками можно найти в S. E. Madnick, *CACM* **10** (1967), 420–424.

В системах обработки списков IPL-V и FLPL не использовались ни сборка мусора, ни технология счетчиков ссылок при работе с разделяемыми списками. Вместо этого каждый список "принадлежал" одному списку и "заимствовался" всеми другими списками, которые на него ссылались. Список удалялся, когда его владелец позволял это. Следовательно, программист должен был сам гарантировать, что при удалении список никому не был одолжен. Технология счетчика ссылок для списков введена Д. Э. Коллинзом (G. E. Collins), *CACM* **3** (1960), 655–657, и позже проанализирована в *CACM* **9** (1966), 578–588. Сборка мусора впервые была описана в статье Мак-Карти в 1960 году [см. также примечания Вейзенбаума в *CACM* **7** (1964), 38, и статью Коэна (Cohen) и Триллинга (Trilling), *BIT* **7** (1967), 22–30].

Рост понимания важности работы со связями естественным образом привел к их включению в алгебраические языки программирования, созданные после 1965 года. Новые языки позволяли программистам выбрать подходящую форму представления данных, не прибегая к ассемблеру и избегая накладных расходов универсальных структур списков. Некоторыми из фундаментальных шагов на этом пути были работы N. Wirth and H. Weber, *CACM* **9** (1966), 13–23, 25, 89–99; H. W. Lawson, *CACM* **10** (1967), 358–367; C. A. R. Hoare, *Symbol Manipulation Languages and Techniques*, ed. by D. G. Bobrow (Amsterdam: North-Holland, 1968), 262–284; O.-J. Dahl and K. Nygaard, *CACM* **9** (1966), 671–678; A. van Wijngaarden, B. J. Mailloux, J. E. L. Peck, and C. H. A. Koster, *Numerische Math.* **14** (1969), 79–218; Dennis M. Ritchie, *History of Programming Languages—II* (ACM Press, 1996), 671–698.

Алгоритмы динамического выделения памяти появились за несколько лет до их описания в печати. Очень интересное обсуждение было подготовлено В. Т. Комфортом (W. T. Comfort) в 1961 году и опубликовано в *CACM* **7** (1964), 357–362. Метод граничного дескриптора, описанный в разделе 2.5, был разработан автором книги в 1962 году для использования в операционной системе компьютера Burroughs B5000. Система двойников впервые использовалась Г. Марковицем (H. Markowitz) в системе программирования SIMSCRIPT в 1963 году, а также была независимо разработана и опубликована К. Ноултоном (K. Knowlton), *CACM* **8** (1965), 623–625 [см. также *CACM* **9** (1966), 616–625]. Дополнительную информацию о динамическом выделении памяти можно найти в статьях Iliffe and Jodeit, *Comp. J.* **5** (1962), 200–209; Bailey, Barnett, and Burleson, *CACM* **7** (1964), 339–346; A. T. Berztiss, *CACM* **8** (1965), 512–513; D. T. Ross, *CACM* **10** (1967), 481–492.

Общее обсуждение информационных структур и их связи с программированием было подготовлено Мэри д'Империо (Mary d'Imperio) — "Data Structures and their Representation in Storage", *Annual Review in Automatic Programming* **5** (Oxford: Pergamon Press, 1969). Ее статья служит серьезным путеводителем по истории предмета, поскольку включает детальный анализ структур, использованных в двенадцати системах обработки списков и работы со строками. Дополнительные исторические детали можно найти в трудах двух симпозиумов, *CACM* **3** (1960), 183–234, и *CACM* **9** (1966), 567–643 (отдельные статьи из этих трудов упоминались ранее).

Великолепно аннотированная библиография ранних работ по манипуляции символами и математическими формулами, имеющая множество связей с материалом этой главы, была составлена Д. Э. Саммет (J. E. Sammet); см. *Computing Reviews* **7** (July–August, 1966), B1–B31.

В настоящей главе очень подробно рассматривались некоторые типы информационных структур, и (чтобы за деревьями суметь рассмотреть лес), вероятно, будет разумно проанализировать, что нами изучено, и подвести краткий итог по теме "информационные структуры" с точки зрения более широкой перспективы. Начав с базовой идеи *узла* (*node*) как элемента данных, можно найти множество примеров, иллюстрирующих удобные пути представления структурных отношений либо неявно (основываясь на относительном порядке, в котором узлы хранятся в памяти компьютера), либо явно (с помощью указывающих на другие узлы связей в узлах). Объем структурной информации, которая должна быть представлена в таблицах компьютерных программ, зависит от операций, которые должны выполняться над узлами.

Из методических соображений мы, в основном, сконцентрировали свое внимание на связях между информационными структурами и их компьютерным представлением, а не рассматривали эти вопросы по отдельности. Однако для углубленного понимания полезно рассмотреть предмет с более абстрактной точки зрения, отделив идеи, которые могут быть изучены самостоятельно. Был разработан ряд достойных внимания подходов такого типа; особо могут быть рекомендованы для ознакомления следующие ранние работы: G. Mealy, "Another look at data", *Proc. AFIPS Fall Joint Computer Conf.* **31** (1967), 525–534; J. Earley, "Toward an understanding of data structures", *CACM* **14** (1971), 617–627; C. A. R. Hoare, "Notes on data structuring", in *Structured Programming* by O.-J. Dahl, E. W. Dijkstra, and C. A. R. Hoare (Academic Press, 1972), 83–174; Robert W. Engles, "A tutorial on data-base organization", *Annual Review in Automatic Programming* **7** (1972), 3–63.

Материал этой главы не охватывает предмет информационных структур во всей полноте. Как минимум не рассматривались три важных вопроса.

a) Часто необходимо найти узел (или множество узлов, обладающих некоторым значением) в таблице. Такая операция нередко оказывает серьезное влияние на структуру таблицы. Этот вопрос детально описывается в главе 6.

b) В основном, нами рассматривалось внутреннее представление структур в компьютере; однако это всего лишь одна часть темы, поскольку структуры должны быть также представлены как внешние входные или выходные данные. В простых случаях с внешними структурами можно работать, по сути, при помощи тех же технологий, которые были здесь рассмотрены. Однако очень важен процесс преобразования строк символов в более сложные структуры и обратно. Такие процессы анализируются в главах 9 и 10.

c) В основном, в главе обсуждалось представление структур данных в высокоскоростной памяти с произвольным доступом. При использовании медленных устройств, таких как диски и ленты, все структурные проблемы обостряются и все более важными становятся эффективные алгоритмы и эффективные схемы представления данных. Узлы, связанные один с другим, в этих случаях должны располагаться в близких областях памяти. Обычно эти проблемы трудно обсуждать в общем виде, так как они сильно зависят от характеристик конкретных машин. Простейшие примеры из этой главы должны помочь читателю подготовиться к решению сложных проблем, возникающих в связи с использованием менее идеальных устройств хранения информации. Некоторые из этих проблем детально обсуждаются в главах 5 и 6.

В чем же заключается смысл рассматриваемых в этой главе тем? Видимо, наиболее важный вывод, который можно сделать, изучив предложенный материал, состоит в том, что рассмотренные идеи не ограничиваются компьютерным программированием; в целом, они вполне применимы в повседневной жизни. Коллекция узлов, содержащих поля, одни из которых указывают на другие узлы, представляется очень хорошей абстрактной моделью структурных отношений любого вида. Такая модель показывает, каким образом можно построить сложные структуры из простых, а соответствующие алгоритмы для работы со структурами могут быть разработаны естественным образом.

Поэтому представляется разумным продолжить разработку теории о связанных множествах узлов по сравнению с тем, что известно в настоящее время. Возможно,

наиболее очевидный путь создания такой теории состоит в определении нового вида абстрактной машины или "автомата" для работы со связанными структурами. Например, подобное устройство может быть неформально определено следующим образом. Имеются числа k, l, r и s, такие, что автомат обрабатывает узлы, содержащие k полей связи и r информационных полей. Он имеет l регистров связи и s информационных регистров, обеспечивающих управление выполняемыми процессами. Информационные поля и регистры могут содержать любые символы из некоторого заданного набора информационных символов. Каждое из полей связи и регистров связи содержит либо Λ, либо указатель на узел. Машина может (i) создавать новые узлы (помещая связи с узлом в регистр), (ii) проверять равенство символов или значений связей и (iii) переносить информационные символы или значения связей между регистрами и узлами. Непосредственно доступны только узлы, на которые указывают регистры связей. Соответствующие ограничения на поведение машины делают ее эквивалентом ряда других видов автоматов.

Аналогичная модель вычислений была предложена А. Н. Колмогоровым в 1952 году. Его машина, по сути, работала с графом G, имеющим специально созданную стартовую вершину v_0. Действия на каждом шаге зависят только от подграфа G', состоящего из всех вершин на расстоянии $\leq n$ от v_0 в G, и заключаются в замещении G' в G другим графом $G'' = f(G')$, где G'' включает v_0 и вершины на расстоянии, в точности равном n от v_0, и, возможно, другие (вновь созданные) вершины. Остаток графа G является неизменным. Здесь n — фиксированное число, определяющееся отдельно для каждого конкретного алгоритма, которое может быть произвольно большим. Каждой вершине назначается символ из некоторого конечного алфавита с тем ограничением, что у вершины не может быть двух соседних вершин с одинаковыми символами. (См. А. Н. Колмогоров, *Успехи мат. наук* **8**, 4 (1953), 175–176; Колмогоров и Успенский, *Успехи мат. наук* **13**, 4 (1958), 3–28; *Amer. Math. Soc. Translations*, series 2, **29** (1963), 217–245.)

Связывающие автоматы могут легко моделировать машины графов, используя ограниченное сверху количество шагов на один шаг работы графа. Напротив, маловероятно, чтобы машины графов могли моделировать произвольные связывающие автоматы без неограниченного увеличения времени работы, если не перейти от неориентированных графов к ориентированным, чтобы работать с вершинами с ограниченной степенью. И, конечно, связывающая модель гораздо ближе к операциям, доступным программисту на реальной машине, в отличие от модели с использованием графа.

Самыми интересными проблемами, которые предстоит решить для такого рода устройств, являются определение скорости решения поставленных ими задач, т. е. подсчет количества узлов, требуемых для решения той или иной задачи (например, для трансляции какого-либо формального языка). В то время, когда автор приступил к работе над настоящей главой, были получены интересные результаты такого рода (отметим работы Ю. Хартманиса (J. Hartmanis) и Р. Э. Стирнса (R. E. Stearns)), но только для специальных классов машин Тьюринга с множеством лент и головок чтения/записи. Модель машины Тьюринга сравнительно нереальна, а потому полученные результаты имеют мало общего с решением практических задач.

Следует признать, что при стремлении количества созданных связывающим автоматом узлов n к бесконечности неизвестно, как построить такое устройство фи-

зически, поскольку желательно, чтобы операции машины выполнялись за одно и то же время независимо от размера n. Если связывание представлено с использованием адресов в машинной памяти, необходимо определить границу для количества узлов, поскольку поля связей имеют фиксированный размер. Многоленточная машина Тьюринга поэтому представляет собой более реалистичную модель при стремлении n к бесконечности. Представляется также обоснованной уверенность в том, что описанные выше связывающие автоматы приведут к созданию более приемлемой теории сложности алгоритмов, чем машина Тьюринга, даже при рассмотрении асимптотических формул для больших n, так как эта теория больше подходит для практических значений n. Кроме того, когда n становится больше, чем 10^{30} или около того, даже одноленточная машина Тьюринга не является реалистичной: она никогда не будет построена. Принципы важнее реалий.

Со времени первого написания автором большинства из приведенных выше комментариев утекло много воды, и можно порадоваться, что в теории связывающих автоматов (сегодня называемых *машинами указателей* (*pointer machines*)) достигнут определенный прогресс, хотя, конечно же, предстоит еще немало сделать в этой области.

Разработаны общие принципы программирования.
Большинство из них давно используется
на станции Канзас-Сортировочная...

— ДЕРРИК ЛЕМЕР (DERRICK LEHMER) (1949)

Я уверен, вы согласитесь со мной ... что если
со страницей 534 мы встречаемся только во второй главе,
то первая глава должна быть невыносимо длинной.

— ШЕРЛОК ХОЛМС (SHERLOCK HOLMES), *Аллея страха* (*The Valley of Fear*) (1888)

ОТВЕТЫ К УПРАЖНЕНИЯМ

Тебе ответом угождать не должен!

— ШЕЙЛОК, *Венецианский купец* (акт IV, сцена 1)

ПРИМЕЧАНИЯ К УПРАЖНЕНИЯМ

1. Средняя задача для читателя с математическими наклонностями.

4. См. W. J. LeVeque, *Topics in Number Theory* **2** (Reading, Mass.: Addison-Wesley, 1956), Chapter 3; P. Ribenboim, *13 Lectures on Fermat's Last Theorem* (New York: Springer-Verlag, 1979); A. Wiles, *Annals of Mathematics* **141** (1995), 443–551.

РАЗДЕЛ 1.1

1. $t \leftarrow a$, $a \leftarrow b$, $b \leftarrow c$, $c \leftarrow d$, $d \leftarrow t$.

2. После первого случая выполнения шага E1 значения переменных m и n — это предыдущие значения n и r соответственно, а $n > r$.

3. Алгоритм F (*Алгоритм Евклида*). Даны два целых положительных числа m и n. Найти их наибольший общий делитель.

F1. [Остаток от деления m/n.] Разделите m на n и пусть остатком будет m.

F2. [Это нуль?] Если $m = 0$, то работа алгоритма завершается и ответом будет n.

F3. [Остаток от деления n/m.] Разделите n на m, и пусть остатком будет n.

F4. [Это нуль?] Если $n = 0$, то работа алгоритма завершается и ответом будет m; в противном случае вернуться к шагу F1. ▮

4. С помощью алгоритма E получаем: $n = 6099$, 2166, 1767, 399, 171, 57. Ответ: 57.

5. Не обладает свойствами конечности, определенности и эффективности и, пожалуй, не имеет выходных данных. Относительно формы записи: отсутствуют буквы перед номерами шагов, краткие описания шагов, а также символ "▮".

6. Применяя алгоритм E для $n = 5$ и $m = 1, 2, 3, 4, 5$, находим, что шаг E1 выполняется 2, 3, 4, 3, 1 раз соответственно. Таким образом, среднее равно $2.6 = T_5$.

7. Во всех случаях, за исключением конечного их числа, $n > m$. А если $n > m$, то в ходе первой итерации алгоритма E эти числа просто меняются местами; поэтому $U_m = T_m + 1$.

8. Пусть $A = \{a, b, c\}$, $N = 5$. Выполнение алгоритма закончится, когда получим строку $a^{\gcd(m,n)}$.

j	θ_j	ϕ_j	b_j	a_j	
0	ab	(Пустая строка)	1	2	Удалить одно a и одно b либо перейти к 2.
1	(Пустая строка)	c	0	0	Добавить c с левого края, перейти к 0.
2	a	b	2	3	Заменить все a на b.
3	c	a	3	4	Заменить все c на a.
4	b	b	0	5	Если b еще остались, повторить.

9. Например, можно сказать, что C_2 представляет C_1, если существуют функция g из I_1 в I_2, функция h из Q_2 в Q_1, переводящая Ω_2 в Ω_1, и функция j из Q_2 в множество положительных целых чисел, удовлетворяющие следующим условиям.

a) Для любого элемента x из множества I_1 метод вычислений C_1 дает выходное значение y для входного x тогда и только тогда, когда существует y' из Ω_2, для которого C_2 дает выходное значение y' для входа $g(x)$ и $h(y') = y$.

b) Для любого q из Q_2 имеет место равенство $f_1(h(q)) = h(f_2^{[j(q)]}(q))$, где $f_2^{[j(q)]}$ — это $j(q)$-я итерация функции f_2.

Например, пусть C_1 — метод вычислений, определенный соотношениями (2), а C_2 определяется соотношениями $I_2 = \{(m, n)\}$, $\Omega_2 = \{(m, n, d)\}$, $Q_2 = I_2 \cup \Omega_2 \cup \{(m, n, a, b, 1)\} \cup \{(m, n, a, b, r, 2)\} \cup \{(m, n, a, b, r, 3)\} \cup \{(m, n, a, b, r, 4)\} \cup \{(m, n, a, b, 5)\}$. Пусть $f_2((m, n)) = (m, n, m, n, 1)$; $f_2((m, n, d)) = (m, n, d)$; $f_2((m, n, a, b, 1)) = (m, n, a, b, a \bmod b, 2)$; $f_2((m, n, a, b, r, 2)) = (m, n, b)$, если $r = 0$, в противном случае $(m, n, a, b, r, 3)$; $f_2((m, n, a, b, r, 3)) = (m, n, b, b, r, 4)$; $f_2((m, n, a, b, r, 4)) = (m, n, a, r, 5)$; $f_2((m, n, a, b, 5)) = f_2((m, n, a, b, 1))$.

Теперь пусть $h((m, n)) = g((m, n)) = (m, n)$; $h((m, n, d)) = (d)$; $h((m, n, a, b, 1)) = (a, b, 0, 1)$; $h((m, n, a, b, r, 2)) = (a, b, r, 2)$; $h((m, n, a, b, r, 3)) = (a, b, r, 3)$; $h((m, n, a, b, r, 4)) = h(f_2((m, n, a, b, r, 4)))$; $h((m, n, a, b, 5)) = (a, b, b, 1)$; $j((m, n, a, b, r, 3)) = j((m, n, a, b, r, 4)) = 2$; в остальных случаях $j(q) = 1$. Тогда C_2 представляет C_1.

Замечания. Конечно, весьма заманчиво попытаться дать более простое определение, например такое. Пусть g отображает Q_1 в Q_2 и должно выполняться только следующее условие: для любой вычисляемой последовательности $x_0, x_1, \ldots$ метода C_1 $g(x_0), g(x_1), \ldots$ является подпоследовательностью вычисляемой последовательности метода C_2, начинающейся с $g(x_0)$. Но это определение неадекватно; в приведенном выше примере в методе C_1 первоначальные значения m и n забываются, а в методе C_2 — нет.

Если C_2 представляет C_1 с помощью функций g, h, j, а C_3 представляет C_2 с помощью функций g', h', j', то C_3 представляет C_1 с помощью функций g'', h'', j'', где

$$g''(x) = g'(g(x)), \qquad h''(x) = h(h'(x)) \qquad \text{и} \qquad j''(q) = \sum_{0 \le k < j(h'(q))} j'(q_k),$$

если $q_0 = q$ и $q_{k+1} = f_3^{[j'(q_k)]}(q_k)$. Следовательно, определенное выше отношение является транзитивным. Если функция j ограничена, то будем говорить, что C_2 *непосредственно представляет* C_1; это отношение также является транзитивным. Отношение "C_2 представляет C_1" порождает отношение эквивалентности, при котором два метода вычислений эквивалентны тогда и только тогда, когда их функции от входных данных являются изоморфными. Отношение "C_2 непосредственно представляет C_1" порождает более интересное отношение эквивалентности, которое, вероятно, отвечает интуитивно понятному представлению о том, что это "в сущности, тот же самый алгоритм".

Об альтернативном подходе к моделированию речь идет в работе R. W. Floyd, R. Beigel, *The Language of Machines* (Computer Science Press, 1994), раздел 3.3.

РАЗДЕЛ 1.2.1

1. (a) Докажите $P(0)$. (b) Докажите, что $P(0), \ldots, P(n)$ влекут за собой $P(n+1)$ для всех $n \geq 0$.

2. Теорема не доказана для $n = 2$. Если во второй части доказательства принять $n = 1$, то придется допустить, что $a^{-1} = 1$. Если это условие выполняется (т. е. $a = 1$), то теорема действительно справедлива.

3. На самом деле правая часть уравнения должна иметь вид $1 - 1/n$. Ошибка возникает при доказательстве случая $n = 1$, когда левую часть нужно считать либо не имеющей смысла, либо равной нулю (поскольку есть $n - 1$ слагаемых).

5. Если n — простое число, то очевидно, что оно является произведением простых чисел. В противном случае n разлагается на множители, т. е. $n = km$ для некоторых k и m, $1 < k, m < n$. Поскольку и k, и m меньше n, по индукции их можно записать как произведение простых чисел. Следовательно, n является произведением простых чисел, фигурирующих в представлениях k и m.

6. Используя обозначения, представленные на рис. 4, докажем, что *A5* влечет *A6*. Это очевидно, так как из *A5* следует, что $(a' - qa)m + (b' - qb)n = (a'm + b'n) - q(am + bn) = c - qd = r$.

7. $n^2 - (n-1)^2 + \cdots - (-1)^n 1^2 = 1 + 2 + \cdots + n = n(n+1)/2$.

8. (a) Покажем, что $(n^2 - n + 1) + (n^2 - n + 3) + \cdots + (n^2 + n - 1)$ равно n^3. Эта сумма равна $(1 + 3 + \cdots + (n^2 + n - 1)) - (1 + 3 + \cdots + (n^2 - n - 1)) = ((n^2 + n)/2)^2 - ((n^2 - n)/2)^2 = n^3$ (мы воспользовались соотношением (2)). Но требовалось дать доказательство по индукции, поэтому нужно использовать другой подход! Для $n = 1$ доказательство очевидно. Пусть $n \geq 1$; так как $(n+1)^2 - (n+1) = n^2 - n + 2n$, то, сравнивая слагаемые сумм для значений параметров $n + 1$ и n, получаем, что $(n+1)$-я сумма равна n-й сумме плюс

$$\underbrace{2n + \cdots + 2n}_{n \text{ раз}} + (n+1)^2 + (n+1) - 1;$$

а это равно $n^3 + 2n^2 + n^2 + 3n + 1 = (n+1)^3$. (b) Поскольку первое слагаемое для $(n+1)$-й суммы на два больше последнего слагаемого n-й суммы, то из соотношения (2) получаем, что $1^3 + 2^3 + \cdots + n^3$ равно сумме последовательных нечетных чисел от 1 до $n^2 + n - 1 = (n(n+1)/2)^2 = (1 + 2 + \cdots + n)^2$.

10. Для $n = 10$ доказательство очевидно. Если $n \geq 10$, то имеем $2^{n+1} = 2 \cdot 2^n > (1 + 1/n)^3 2^n$, а по предположению индукции это больше, чем $(1 + 1/n)^3 n^3 = (n+1)^3$.

11. $(-1)^n (n+1)/(4(n+1)^2 + 1)$.

12. Единственным нетривиальным моментом такого обобщения является вычисление целого числа q на шаге E2. Это можно сделать путем повторного вычитания, сводя задачу к выяснению, является ли $u + v\sqrt{2}$ положительным, отрицательным или равным нулю, что уже не представляет особых проблем.

Легко показать, что если $u + v\sqrt{2} = u' + v'\sqrt{2}$, то $u = u'$ и $v = v'$, так как $\sqrt{2}$ — иррациональное число. Теперь ясно, что 1 и $\sqrt{2}$ не имеют общего делителя, если мы определим делимость в следующем смысле: $u + v\sqrt{2}$ делит $a(u + v\sqrt{2})$ тогда и только тогда, когда a — целое число. Алгоритм, обобщенный подобным образом, вычисляет регулярную цепную дробь отношений его входов (см. раздел 4.5.3).

[*Замечание.* Расширим понятие делимости следующим образом: будем говорить, что $u + v\sqrt{2}$ делит $a(u + v\sqrt{2})$ тогда и только тогда, когда a имеет вид $u' + v'\sqrt{2}$, где u' и v' — целые числа. В этом случае *существует* способ обобщить алгоритм E таким образом, чтобы он всегда был конечным. Действительно, если на шаге E2 мы имеем $c = u + v\sqrt{2}$

и $d = u' + v'\sqrt{2}$, то получаем $c/d = c(u' - v'\sqrt{2})/(u'^2 - 2v'^2) = x + y\sqrt{2}$, где x и y — рациональные числа. Пусть теперь $q = u'' + v''\sqrt{2}$, где u'' и v'' — это ближайшие к x и y целые числа, и положим $r = c - qd$. Если $r = u''' + v'''\sqrt{2}$, то $|u'''^2 - 2v'''^2| < |u'^2 - 2v'^2|$. Следовательно, выполнение алгоритма завершится через конечное число шагов. Более подробную информацию об этом можно найти в учебниках по теории чисел, в разделах, посвященных квадратичным евклидовым областям.]

13. Добавьте "$T \le 3(n-d)+k$" к утверждениям *A3*, *A4*, *A5*, *A6*, где k принимает значения 2, 3, 3, 1 соответственно. Добавьте также "$d > 0$" к утверждению *A4*.

15. (a) Положим $A = S$ в (iii); отсюда следует, что любое непустое вполне упорядоченное множество имеет минимальный (или наименьший) элемент.

(b) Пусть $x \prec y$, если $|x| < |y|$ или если $|x| = |y|$ и $x < 0 < y$.

(c) Нет, так как подмножество всех действительных положительных чисел не удовлетворяет условию (iii). [*Замечание.* Пользуясь так называемой аксиомой выбора, можно дать довольно сложное доказательство того, что любое множество можно вполне упорядочить каким-то образом; но до сих пор никто не определил явным образом отношение, которое вполне упорядочивает множество действительных чисел.]

(d) Чтобы доказать для T_n свойство (iii), воспользуемся индукцией по n. Пусть A — непустое подмножество T_n; рассмотрим A_1, которое представляет собой множество первых компонент элементов A. Так как A_1 — это непустое подмножество S, а S вполне упорядочено, то A_1 содержит наименьший элемент x. Теперь рассмотрим A_x, которое является подмножеством тех элементов из A, первая компонента которых равна x; A_x можно считать подмножеством T_{n-1}, если у всех его элементов изъять первую компоненту. Поэтому по индукции A_x содержит наименьший элемент $(x, x_2, \ldots, x_n)$, который на самом деле является наименьшим элементом A.

(e) Нет, хотя свойства (i) и (ii) выполняются. Если S содержит по меньшей мере два различных элемента, $a \prec b$, то множество (b), (a, b), (a, a, b), (a, a, a, b), (a, a, a, a, b), ... не имеет наименьшего элемента. С другой стороны, T можно вполне упорядочить, если определить в T_n отношение $(x_1, \ldots, x_m) \prec (y_1, \ldots, y_n)$ при $m < n$ или $(x_1, \ldots, x_n) \prec (y_1, \ldots, y_n)$ при $m = n$.

(f) Пусть множество S вполне упорядочено отношением $\prec$. Если такая бесконечная последовательность существует, то множество A, содержащее члены этой последовательности, не будет удовлетворять свойству (iii), так как ни один элемент этой последовательности не может быть наименьшим. Обратно, пусть $\prec$ — отношение, удовлетворяющее условиям (i) и (ii), но не (iii), и пусть A — непустое подмножество S, не имеющее наименьшего элемента. Так как A — непустое, мы можем найти x_1 из A; так как x_1 не является наименьшим элементом A, то существует x_2 из A, для которого $x_2 \prec x_1$; так как x_2 тоже не является наименьшим элементом, мы можем найти $x_3 \prec x_2$ и т. д.

(g) Пусть A — множество всех x, для которых утверждение $P(x)$ ложно. Если A не пусто, то оно содержит наименьший элемент x_0. Следовательно, $P(y)$ верно для всех $y \prec x_0$. Но отсюда вытекает, что $P(x_0)$ верно, следовательно, x_0 не принадлежит A (получаем противоречие). Таким образом, A должно быть пустым, т. е утверждение $P(x)$ всегда верно.

РАЗДЕЛ 1.2.2

1. Такого числа нет; для любого положительного рационального числа r можно указать меньшее число, например $r/2$.

2. Нет, если подряд идет бесконечно много девяток; в этом случае согласно соотношению (2) десятичным представлением данного действительного числа будет $1 + .24000000\ldots$.

3. $-1/27$, но в тексте раздела операция возведения в степень отрицательных чисел не определена.

4. 4.

6. Для любого действительного числа существует единственное десятичное представление, поэтому $x = y$ тогда и только тогда, когда $m = n$ и $d_i = e_i$ для всех $i \geq 1$. Если $x \neq y$, то нужно сравнивать m с n, d_1 с e_1, d_2 с e_2 и т. д.; когда обнаружится первое неравенство, бóльшая цифра будет принадлежать большему из чисел $\{x, y\}$.

7. Можно воспользоваться индукцией по x и сначала доказать эти правила для положительного, а затем для отрицательного x. Детали доказательства мы здесь опускаем.

8. Испытывая последовательно $n = 0, 1, 2, \ldots$, находим значение n, для которого $n^m \leq u < (n+1)^m$. Предполагая по индукции, что $n, d_1, \ldots, d_{k-1}$ уже определены, находим цифру d_k из условия

$$\left(n + \frac{d_1}{10} + \cdots + \frac{d_k}{10^k}\right)^m \leq u < \left(n + \frac{d_1}{10} + \cdots + \frac{d_k}{10^k} + \frac{1}{10^k}\right)^m.$$

9. $((b^{p/q})^{u/v})^{qv} = (((b^{p/q})^{u/v})^v)^q = ((b^{p/q})^u)^q = ((b^{p/q})^q)^u = b^{pu}$, следовательно, $(b^{p/q})^{u/v} = b^{pu/qv}$. Таким образом, второе правило доказано. Теперь на основе второго правила докажем первое: $b^{p/q}b^{u/v} = (b^{1/qv})^{pv}(b^{1/qv})^{qu} = (b^{1/qv})^{pv+qu} = b^{p/q+u/v}$.

10. Если $\log_{10} 2 = p/q$, где p и q положительны, то $2^q = 10^p$, а это невозможно, так как правая часть уравнения делится на 5, а левая нет.

11. Бесконечно много! Независимо от того, сколько дано цифр десятичного представления числа x, мы все равно не знаем, будет ли $10^x = 1.99999\ldots$ или $2.00000\ldots$, если цифры x соответствуют цифрам $\log_{10} 2$. И в этом нет ничего таинственного или парадоксального; аналогичная ситуация возникает при сложении, например, $.444444\ldots$ с $.55555\ldots$.

12. Существует единственный набор значений $d_1, \ldots, d_8$, удовлетворяющий соотношению (7).

13. (a) Сначала докажите по индукции, что если $y > 0$, то $1+ny \leq (1+y)^n$. Затем положите $y = x/n$ и извлеките из обеих частей неравенства корни n-й степени. (b) $x = b-1$, $n = 10^k$.

14. Во втором равенстве (5) положите $x = \log_b c$, а затем прологарифмируйте обе части.

15. Перенесите $\log_b y$ в другую часть равенства и воспользуйтесь формулой (11).

16. $\ln x / \ln 10$.

17. 5; 1; 1; 0; не определен.

18. Нет, $\log_8 x = \lg x / \lg 8 = \frac{1}{3} \lg x$.

19. Да, так как $\lg n < (\log_{10} n)/.301 < 14/.301 < 47$.

20. Да, это взаимно обратные значения.

21. $(\ln\ln x - \ln\ln b)/\ln b$.

22. Из таблиц приложения A получаем, что $\lg x \approx 1.442695 \ln x$; $\log_{10} x \approx .4342945 \ln x$. Относительная погрешность составляет $\approx (1.442695 - 1.4342945)/1.442695 \approx 0.582\%$.

23. Возьмите фигуру площадью $\ln y$ и разделите ее высоту на x, в то же время умножив ее длину на x. В результате этого преобразования площадь не изменится и будет равна площади фигуры, которая останется после удаления $\ln x$ из $\ln xy$, так как высота в точке $x + xt$ на графике $\ln xy$ равна $1/(x + xt) = (1/(1+t))/x$.

24. Везде вместо 10 подставьте 2.

25. Обратите внимание, что $z = 2^{-p}\lfloor 2^{p-k}x \rfloor$, где p — это точность (т. е. количество двоичных цифр после двоичной точки)*. Величина $y + \log_b x$ приблизительно остается постоянной.

27. Докажите индукцией по k, что

$$x^{2^k}(1-\delta)^{2^{k+1}-1} \le 10^{2^k(n+b_1/2+\cdots+b_k/2^k)}x_k' \le x^{2^k}(1+\epsilon)^{2^{k+1}-1},$$

и прологарифмируйте все части неравенства.

28. В приведенном ниже алгоритме используются те же вспомогательные таблицы, что и раньше (см. выше).

E1. [Инициализация.] Если $1-\epsilon$ — это наибольшее возможное значение x, присвойте $y \leftarrow$ (ближайшее приближенное значение $b^{1-\epsilon}$), $x \leftarrow 1-\epsilon$, $k \leftarrow 1$. (При выполнении следующих шагов величина yb^{-x} останется приблизительно постоянной.)

E2. [Проверка окончания.] Если $x = 0$, то прекратите выполнение.

E3. [Сравнение.] Если $x < \log_b(2^k/(2^k-1))$, то увеличьте k на 1 и повторите этот шаг.

E4. [Замещение значений.] Присвойте $x \leftarrow x - \log_b(2^k/(2^k-1))$, $y \leftarrow y -$ (y сдвигается вправо на k) и перейдите к шагу E2. ▮

Если на шаге E1 положить y равным $b^{1-\epsilon}(1+\epsilon_0)$, то в результате возникает погрешность вычислений при выполнении операций $x \leftarrow x + \log_b(1-2^{-k}) + \delta_j$ и $y \leftarrow y(1-2^{-k})(1+\epsilon_j)$ во время j-го выполнения шага E4, причем δ_j и ϵ_j — некоторые малые погрешности. Когда выполнение алгоритма прекратится, будет вычислено значение $y = b^{x-\Sigma\delta_j}\prod_j(1+\epsilon_j)$. Дальнейший анализ проводится в зависимости от величины b и размера компьютерного слова. Обратите внимание, что и в данном случае, и в упр. 26 можно несколько уточнить оценки ошибок, если рассматривать логарифмы по основанию e. Дело в том, что для большинства значений k табличное значение $\ln(2^k/(2^k-1))$ можно получить с большей точностью: оно равняется $2^{-k} + \frac{1}{2}2^{-2k} + \frac{1}{3}2^{-3k} + \cdots$.

Замечание. Аналогичные алгоритмы можно получить и для тригонометрических функций; см. J. E. Meggitt, *IBM J. Res. and Dev.* **6** (1962), 210–226; **7** (1963), 237–245. См. также T. C. Chen, *IBM J. Res. and Dev.* **16** (1972), 380–388; V. S. Linsky, *Vychisl. Mat.* **2** (1957), 90–119 (В. С. Линский, *Вычисл. мат.* **2** (1957), 90–119); D. E. Knuth, *METAFONT: The Program* (Reading, Mass.: Addison-Wesley, 1986), §120–147.

29. e; 3; 4.

РАЗДЕЛ 1.2.3

1. $a_1 + a_2 + a_3$.

2. $\frac{1}{1} + \frac{1}{3} + \frac{1}{5} + \frac{1}{7} + \frac{1}{9} + \frac{1}{11}$; $\frac{1}{9} + \frac{1}{3} + \frac{1}{1} + \frac{1}{3} + \frac{1}{9}$.

3. Нарушено условие, налагаемое на $p(j)$; с одной стороны, значение $n^2 = 3$ не принимается ни для одного n, а с другой стороны, значение $n^2 = 4$ принимается для *двух* n. [См. соотношение (18).]

4. $(a_{11}) + (a_{21} + a_{22}) + (a_{31} + a_{32} + a_{33}) = (a_{11} + a_{21} + a_{31}) + (a_{22} + a_{32}) + (a_{33})$.

5. Для доказательства достаточно воспользоваться правилом $a\sum_{R(i)} x_i = \sum_{R(i)}(ax_i)$:

$$\left(\sum_{R(i)} a_i\right)\left(\sum_{S(j)} b_j\right) = \sum_{R(i)} a_i\left(\sum_{S(j)} b_j\right) = \sum_{R(i)}\left(\sum_{S(j)} a_i b_j\right).$$

7. Возьмем одну из сумм (например, первую) и запишем по формуле (3). Затем поменяем пределы местами, перенесем из одного в другой члены $a_0, \ldots, a_c$ и по формуле получим вторую сумму.

* Точка, разделяющая целую и дробную части двоичного представления числа. — *Прим. перев.*

8. Пусть для всех $i \geq 0$ $a_{(i+1)i} = +1$, $a_{i(i+1)} = -1$, а все остальные a_{ij} равны нулю; пусть $R(i) = S(i) =$ "$i \geq 0$". Тогда в левой части получим -1, а в правой — $+1$.

9, 10. Нет; правило (d) применимо только для случаев, когда $n \geq 0$. (Для $n = -1$ формула верна, но доказательство — нет.)

11. $(n+1)a$.

12. $\frac{7}{6}(1 - 1/7^{n+1})$.

13. $m(n-m+1) + \frac{1}{2}(n-m)(n-m+1)$; или $\frac{1}{2}(n(n+1) - m(m-1))$.

14. $\frac{1}{4}(n(n+1) - m(m-1))(s(s+1) - r(r-1))$, если $m \leq n$ и $r \leq s$.

15, 16. Ключевые моменты:

$$\sum_{0\leq j\leq n} jx^j = x \sum_{1\leq j\leq n} jx^{j-1} = x \sum_{0\leq j\leq n-1} (j+1)x^j$$
$$= x \sum_{0\leq j\leq n} jx^j - nx^{n+1} + x \sum_{0\leq j\leq n-1} x^j.$$

17. Числу элементов множества S.

18. $S'(j) =$ "$1 \leq j < n$". $R'(i,j) =$ "n кратно i и $i > j$".

19. $a_n - a_{m-1}$.

20. $(b-1)\sum_{k=0}^{n}(n-k)b^k + n + 1 = \sum_{k=0}^{n} b^k$; эта формула следует из формулы (14) и результата упр. 16.

21. $\sum_{R(j)} a_j + \sum_{S(j)} a_j = \sum_j a_j[R(j)] + \sum_j a_j[S(j)] = \sum_j a_j([R(j)] + [S(j)])$; теперь воспользуйтесь тем, что $[R(j)]+[S(j)] = [R(j)$ или $S(j)]+[R(j)$ и $S(j)]$. В целом, обозначение Айверсона позволяет выполнять операции "в строку", а не "под строкой".

22. Для (5) и (7) достаточно просто заменить знак $\sum$ знаком $\prod$. Кроме того, имеем $\prod_{R(i)} b_i c_i = (\prod_{R(i)} b_i)(\prod_{R(i)} c_i)$ и

$$\Big(\prod_{R(j)} a_j\Big)\Big(\prod_{S(j)} a_j\Big) = \Big(\prod_{R(j) \text{ или } S(j)} a_j\Big)\Big(\prod_{R(j) \text{ и } S(j)} a_j\Big).$$

23. Потому что $0+x = x$ и $1 \cdot x = x$. Это позволяет упростить многие операции и равенства, например правило (d) и его аналог из предыдущего упражнения.

25. Первый и последний шаги выполнены верно. На втором шаге i используется для двух различных целей одновременно. А третий шаг, вероятно, должен выглядеть так: $\sum_{i=1}^{n} n$.

26. Приведем ключевые этапы доказательства:

$$\prod_{i=0}^{n}\Big(\prod_{j=0}^{n} a_i a_j\Big) = \prod_{i=0}^{n}\Big(a_i^{n+1}\prod_{j=0}^{n} a_j\Big)$$
$$= \Big(\prod_{i=0}^{n} a_i^{n+1}\Big)\Big(\prod_{i=0}^{n}\Big(\prod_{j=0}^{n} a_j\Big)\Big) = \Big(\prod_{i=0}^{n} a_i\Big)^{2n+2}$$

Тогда искомое произведение равно $(\prod_{i=0}^{n} a_i)^{n+2}$.

28. $(n+1)/2n$.

29. (a) $\sum_{0\leq k\leq j\leq i\leq n} a_i a_j a_k$. (b) Пусть $S_r = \sum_{i=0}^{n} a_i^r$. Тогда для исходной суммы получаем следующее выражение: $\frac{1}{3}S_3 + \frac{1}{2}S_1S_2 + \frac{1}{6}S_1^3$. Более общее решение данной задачи (для большего числа индексов) можно найти в разделе 1.2.9; см. (38).

30. Запишем левую часть в виде $\sum_{1\le j,k\le n} a_j b_k x_j y_k$ и выполним аналогичные преобразования в правой части. (Это тождество является частным случаем более общей формулы из упр. 46 для $m = 2$.)

31. Если положить $a_j = u_j$, $b_j = 1$, $x_j = v_j$ и $y_j = 1$, то получится, что исходная сумма равна $n\sum_{j=1}^n u_j v_j - (\sum_{j=1}^n u_j)(\sum_{j=1}^n v_j)$.

33. Это можно доказать индукцией по n, если переписать формулу в виде

$$\frac{1}{x_n - x_{n-1}}\left(\sum_{j=1}^{n} \frac{x_j^r(x_j - x_{n-1})}{\prod_{1\le k\le n,\, k\ne j}(x_j - x_k)} - \sum_{j=1}^{n} \frac{x_j^r(x_j - x_n)}{\prod_{1\le k\le n,\, k\ne j}(x_j - x_k)}\right).$$

Теперь каждая из этих сумм имеет вид первоначальной суммы, но без $(n-1)$-го элемента в знаменателе. По индукции предполагаем, что искомая формула верна для $0 \le r \le n-1$. А для $r = n$ рассмотрим тождество

$$0 = \sum_{j=1}^{n} \frac{\prod_{k=1}^n (x_j - x_k)}{\prod_{1\le k\le n,\, k\ne j}(x_j - x_k)} = \sum_{j=1}^{n} \frac{x_j^n - (x_1 + \cdots + x_n)x_j^{n-1} + P(x_j)}{\prod_{1\le k\le n,\, k\ne j}(x_j - x_k)},$$

где $P(x_j)$ — полином степени $n-2$; так как по индукции мы предположили, что формула верна для $r = 0, 1, \ldots, n-1$, отсюда получаем, что она верна и для $r = n$.

Замечание. На самом деле д-ра Матрицу опередил Л. Эйлер (L. Euler), который написал о своем открытии Христиану Гольдбаху (Christian Goldbach) 9 ноября 1762 года. См. работу Эйлера *Institutionum Calculi Integralis* **2** (1769), §1169, а также E. Waring, *Phil. Trans.* **69** (1779), 64–67. Приведенный ниже альтернативный метод доказательства, в котором используется теория комплексного переменного, не так прост, но более изящен. По теореме о вычетах значение заданной суммы равно

$$\frac{1}{2\pi i}\int_{|z|=R} \frac{z^r\,dz}{(z - x_1)\ldots(z - x_n)},$$

где $R > |x_1|, \ldots, |x_n|$. Разложение Лорана подынтегральной функции равномерно сходится при $|z| = R$ и выглядит следующим образом:

$$z^{r-n}\left(\frac{1}{1 - x_1/z}\right)\cdots\left(\frac{1}{1 - x_n/z}\right)$$
$$= z^{r-n} + (x_1 + \cdots + x_n)z^{r-n-1} + (x_1^2 + x_1x_2 + \cdots)z^{r-n-2} + \cdots.$$

При почленном интегрировании пропадет все, кроме члена с коэффициентом z^{-1}. Этот метод позволяет получить *общую формулу* для произвольного целого $r \ge 0$:

$$\sum_{\substack{j_1 + \cdots + j_n = r-n+1 \\ j_1, \ldots, j_n \ge 0}} x_1^{j_1}\ldots x_n^{j_n}.$$

[J. J. Sylvester, *Quart. J. Math.* **1** (1857), 141–152.]

34. Если читатель упорно старался решить эту задачу, *не* заглядывая в ответ, то, вероятно, цель упражнения достигнута. Трудно преодолеть искушение рассмотреть числители как многочлены по x, а не по k. И, без сомнения, намного проще доказать значительно более общее соотношение

$$\sum_{k=1}^{n} \frac{\prod_{1\le r\le n-1}(y_k - z_r)}{\prod_{1\le r\le n,\, r\ne k}(y_k - y_r)} = 1,$$

которое является тождеством для $2n-1$ переменных!

35. Если $R(j)$ не выполняется никогда, то $\sup_{R(j)} a_j$ равен $-\infty$. В основе сформулированного аналога правила (a) лежит тождество $a + \max(b, c) = \max(a + b, a + c)$. Аналогично, если все a_i, b_j *неотрицательны*, то имеем

$$\sup_{R(i)} a_i \sup_{S(j)} b_j = \sup_{R(i)} \sup_{S(j)} a_i b_j.$$

Правила (b) и (c) остаются неизменными, а для правила (d) получаем более простую форму

$$\sup(\sup_{R(j)} a_j, \sup_{S(j)} a_j) = \sup_{R(j) \text{ or } S(j)} a_j.$$

36. Вычитаем первый столбец из столбцов с номерами $2, \ldots, n$. К первой строке добавим строки с номерами $2, \ldots, n$. Теперь остается вычислить определитель треугольной матрицы.

37. Вычитаем первый столбец из столбцов с номерами $2, \ldots, n$. Затем вычитаем $(k-1)$-ю строку, умноженную на x_1, из k-й строки, где $k = n, n-1, \ldots, 2$ (именно в этом порядке) Теперь вынесем коэффициент x_1 из первого столбца, а коэффициенты $x_k - x_1$ — из столбцов $k = 2, \ldots, n$. В результате получим определитель матрицы Вандермонда порядка $n-1$, умноженный на $x_1(x_2 - x_1) \ldots (x_n - x_1)$. Далее проводим доказательство по индукции.

Приведем альтернативный метод доказательства с использованием высшей математики. Искомый определитель является многочленом от переменных $x_1, \ldots, x_n$ общей степени $1 + 2 + \cdots + n$. Он обращается в нуль, если $x_j = 0$ или $x_i = x_j$ $(i < j)$ и коэффициент при $x_1^1 x_2^2 \ldots x_n^n$ равен $+1$. Это характерные особенности данного определителя Вообще, если две строки матрицы становятся равными при $x_i = x_j$, то их разность обычно делится на $x_i - x_j$, и этот факт часто помогает ускорить процесс вычисления определителей.

38. Вычитаем первый столбец из столбцов с номерами $2, \ldots, n$ и выносим из второго столбца $(y_1 - y_2)$, из третьего — $(y_1 - y_3)$ и т. д. и из n-го — $(y_1 - y_n)$. После этого выносим $(x_1 + y_1)^{-1}$ из первой, $(x_2 + y_1)^{-1}$ — из второй и т. д. и $(x_n + y_1)^{-1}$ — из n-й строки. Теперь вычитаем первую строку из строк с номерами $2, \ldots, n$ и выносим $(x_1 - x_2)$ $\ldots (x_1 - x_n)(x_1 + y_2)^{-1} \ldots (x_1 + y_n)^{-1}$. В результате остается определитель матрицы Коши порядка $n-1$.

39. Пусть I — это единичная матрица (δ_{ij}), а J — матрица, состоящая из одних единиц. Так как $J^2 = nJ$, имеем $(xI + yJ)((x + ny)I - yJ) = x(x + ny)I$.

40. $\displaystyle\sum_{t=1}^{n} b_{it} x_j^t = x_j \prod_{\substack{1 \le k \le n \\ k \ne i}} (x_k - x_j) \Big/ x_i \prod_{\substack{1 \le k \le n \\ k \ne i}} (x_k - x_i) = \delta_{ij}.$

41. Это немедленно следует из формулы, выражающей элементы обратной матрицы через алгебраические дополнения. Будет интересно привести здесь также прямое доказательство этого факта. Имеем

$$\sum_{t=1}^{n} \frac{1}{x_i + y_t} b_{tj} = \sum_{t=1}^{n} \frac{\prod_{k \ne t}(x_j + y_k - x) \prod_{k \ne i}(x_k + y_t)}{\prod_{k \ne j}(x_j - x_k) \prod_{k \ne t}(y_t - y_k)}$$

при $x = 0$. Правая часть — это многочлен от x, степень которого не превышает $n-1$. Если положить $x = x_j + y_s$, $1 \le s \le n$, то все члены обратятся в нуль, за исключением случая, когда $s = t$ Поэтому значение данного многочлена равно

$$\prod_{k \ne i}(-x_k - y_s) \Big/ \prod_{k \ne j}(x_j - x_k) = \prod_{k \ne i}(x_j - x_k - x) \Big/ \prod_{k \ne j}(x_j - x_k).$$

Значения таких многочленов степени, не превышающей $n-1$, совпадают в n различных точках x, поэтому они совпадают и при $x=0$. Следовательно,

$$\sum_{t=1}^{n}\frac{1}{x_i+y_t}b_{tj}=\prod_{k\neq i}(x_j-x_k)\Big/\prod_{k\neq j}(x_j-x_k)=\delta_{ij}.$$

42. $n/(x+ny)$.

43. $1-\prod_{k=1}^{n}(1-1/x_k)$. Это легко проверить, если какое-либо значение x_i равно 1, так как обратная матрица к любой матрице, строка либо столбец которой состоит из одних единиц, должна содержать элементы, сумма которых равна 1. Если ни одно из x_i не равно единице, просуммируем элементы i-й строки, как в упр. 44, и получим $\prod_{k\neq i}(x_k-1)/x_i\prod_{k\neq i}(x_k-x_i)$. Теперь можно просуммировать это выражение по i, используя упр. 33 для $r=0$ (умножая числитель и знаменатель на (x_i-1)).

44. Применив упр. 33, находим

$$c_j=\sum_{i=1}^{n}b_{ij}=\prod_{k=1}^{n}(x_j+y_k)\Big/\prod_{\substack{1\le k\le n\\ k\neq j}}(x_j-x_k).$$

Поэтому

$$\sum_{j=1}^{n}c_j=\sum_{j=1}^{n}\frac{(x_j^n+(y_1+\cdots+y_n)x_j^{n-1}+\cdots)}{\prod_{1\le k\le n,\,k\neq j}(x_j-x_k)}$$

$$=(x_1+x_2+\cdots+x_n)+(y_1+y_2+\cdots+y_n).$$

45. Пусть $x_i=i$, $y_j=j-1$. Из упр. 44 следует, что сумма элементов обратной матрицы равна $(1+2+\cdots+n)+((n-1)+(n-2)+\cdots+0)=n^2$. А из упр. 38 получаем, что элементами обратной матрицы являются

$$b_{ij}=\frac{(-1)^{i+j}(i+n-1)!\,(j+n-1)!}{(i+j-1)(i-1)!^2(j-1)!^2(n-i)!\,(n-j)!}.$$

Это выражение можно записывать различными способами с помощью биномиальных коэффициентов, например

$$\frac{(-1)^{i+j}ij}{i+j-1}\binom{-i}{n}\binom{n}{i}\binom{-j}{n}\binom{n}{j}=(-1)^{i+j}j\binom{i+j-2}{i-1}\binom{i+n-1}{i-1}\binom{j+n-1}{n-i}\binom{n}{j}.$$

Из последней формулы видно не только то, что b_{ij} является простым числом, но и то, что оно делится на i, j, n, $i+j-1$, $i+n-1$, $j+n-1$, $n-i+1$ и $n-j+1$. Но, пожалуй, самой удачной формулой для b_{ij} является

$$(i+j-1)\binom{i+j-2}{i-1}^2\binom{-(i+j)}{n-i}\binom{-(i+j)}{n-j}.$$

Нам было бы чрезвычайно трудно решить данную задачу, если бы мы не поняли, что матрица Гильберта — это частный случай матрицы Коши. Оказывается, более общую задачу решить намного проще, чем ее частный случай! Часто имеет смысл обобщить задачу до ее "индуктивного замыкания", т. е. до минимального обобщения, которое включало бы в себя все частные случаи, возникающие при попытке провести доказательство по индукции. В данном случае алгебраические дополнения элементов матрицы Коши тоже являются матрицами Коши, но алгебраические дополнения элементов матрицы Гильберта не являются матрицами Гильберта. [Более подробную информацию об этом можно найти в J. Todd, *J. Res. Nat. Bur. Stand.* **65** (1961), 19–22.]

46. Для любых целых $k_1, k_2, \ldots, k_m$ положим $\epsilon(k_1, \ldots, k_m) = \operatorname{sign}(\prod_{1\le i<j\le m}(k_j - k_i))$, где $\operatorname{sign} x = [x>0] - [x<0]$. Если $(l_1, \ldots, l_m)$ получается из $(k_1, \ldots, k_m)$ в результате перестановки значений k_i и k_j, то имеем $\epsilon(l_1, \ldots, l_m) = -\epsilon(k_1, \ldots, k_m)$. Отсюда следует равенство $\det(B_{k_1\,\ldots k_m}) = \epsilon(k_1, \ldots, k_m)\det(B_{j_1\ldots j_m})$, если $j_1 \le \cdots \le j_m$ — это числа $k_1, \ldots, k_m$, записанные в порядке неубывания. Теперь из определения детерминанта получаем, что

$$\begin{aligned}
\det(AB) &= \sum_{1\le l_1,\ldots,l_m\le m} \epsilon(l_1,\ldots,l_m)\left(\sum_{k=1}^{n} a_{1k}b_{kl_1}\right)\ldots\left(\sum_{k=1}^{n} a_{mk}b_{kl_m}\right)\\
&= \sum_{1\le k_1,\ldots,k_m\le n} a_{1k_1}\ldots a_{mk_m} \sum_{1\le l_1,\ldots,l_m\le m} \epsilon(l_1,\ldots,l_m) b_{k_1l_1}\ldots b_{k_ml_m}\\
&= \sum_{1\le k_1,\ldots,k_m\le n} a_{1k_1}\ldots a_{mk_m}\det(B_{k_1\,\ldots k_m})\\
&= \sum_{1\le k_1,\ldots,k_m\le m} \epsilon(k_1,\ldots,k_m) a_{1k_1}\ldots a_{mk_m}\det(B_{j_1\,\ldots j_m})\\
&= \sum_{1\le j_1\le\cdots\le j_m\le n} \det(A_{j_1\,\ldots j_m})\det(B_{j_1\ldots j_m}).
\end{aligned}$$

И наконец, если два индекса j равны, то $\det(A_{j_1\ldots j_m}) = 0$. [*J. de l'École Polytechnique* **9** (1813), 280–354; **10** (1815), 29–112. Бине и Коши представили свои статьи в один и тот же день 1812 года.]

47. Пусть $a_{ij} = (\prod_{k=1}^{j-1}(x_i+p_k))(\prod_{k=j+1}^{n}(x_i+q_k))$. Вычтем $(k-1)$-й столбец из k-го столбца и вынесем $p_{k-j} - q_k$, где $k = n, n-1, \ldots, j+1$ (именно в этом порядке), а $j = 1, 2, \ldots, n-1$ (именно в этом порядке). После вынесения останется произведение $\prod_{1\le i<j\le n}(p_i - q_j)$, умноженное на $\det(b_{ij})$, где $b_{ij} = \prod_{k=j+1}^{n}(x_i + q_k)$. Теперь из k-го столбца вычтем $(k+1)$-й столбец, умноженный на q_{k+j}, где $k = 1, \ldots, n-j$, а $j = 1, \ldots, n-1$. В результате получим $\det(c_{ij})$, где $c_{ij} = x_i^{n-j}$, что, в сущности, определяет матрицу Вандермонда. Продолжая преобразования, как в упр. 37, т. е. выполняя операции уже не над столбцами, а над строками, получим

$$\det(a_{ij}) = \prod_{1\le i<j\le n}(x_i - x_j)(p_i - q_j).$$

Для $p_j = q_j = y_j$, где $1 \le j \le n$, матрица из этого упражнения является матрицей Коши, i-я строка которой умножена на $\prod_{j=1}^{n}(x_i+y_j)$. Поэтому данный результат обобщает упр. 38, так как добавляется $n-1$ независимых параметров. [*Manuscripta Math.* **69** (1990), 177–178.]

РАЗДЕЛ 1.2.4

1. 1, −2, −1, 0, 5.

2. $\lfloor x \rfloor$.

3. По определению $\lfloor x \rfloor$ — наибольшее целое число, меньшее или равное x, поэтому $\lfloor x \rfloor$ — такое целое, для которого $\lfloor x \rfloor \le x$ и $\lfloor x \rfloor + 1 > x$. Последние свойства и тот факт, что для целых чисел m и n неравенство $m < n$ выполняется тогда и только тогда, когда $m \le n-1$, позволяют легко доказать пп. (a) и (b). Для доказательства пп. (c) и (d) используются аналогичные рассуждения. И наконец, пп. (e) и (f) — это просто комбинации предыдущих утверждений.

4. Так как $x - 1 < \lfloor x \rfloor \le x$, то $-x + 1 > -\lfloor x \rfloor \ge -x$, откуда следует нужный результат.

5. $\lfloor x + \frac{1}{2} \rfloor$. Значение ($-x$ округленное) будет равно значению $-(x$ округленное), *за исключением случая*, когда $x \bmod 1 = \frac{1}{2}$. В последнем случае отрицательные значения

округляются по направлению к нулю, а положительные — в противоположном от нуля направлении.

6. Утверждение (a) верно: $\lfloor\sqrt{x}\rfloor = n \iff n^2 \le x < (n+1)^2 \iff n^2 \le \lfloor x\rfloor < (n+1)^2 \iff \lfloor\sqrt{\lfloor x\rfloor}\,\rfloor = n$. Аналогично доказывается справедливость утверждения (b). Но утверждение (c) неверно, например, для $x = 1.1$.

7. $\lfloor x+y\rfloor = \lfloor\lfloor x\rfloor + x \bmod 1 + \lfloor y\rfloor + y \bmod 1\rfloor = \lfloor x\rfloor + \lfloor y\rfloor + \lfloor x \bmod 1 + y \bmod 1\rfloor$. Для функции "потолок" выполняется неравенство со знаком "$\ge$", причем равенство достигается тогда и только тогда, когда либо x или y — целое, либо $x \bmod 1 + y \bmod 1 > 1$.

8. 1, 2, 5, -100.

9. -1, 0, -2.

10. 0.1, 0.01, -0.09.

11. $x = y$.

12. Все.

13. $+1$, -1.

14. 8.

15. Умножаем обе части равенства (1) на z; при $y = 0$ доказательство очевидно.

17. В качестве примера рассмотрим свойство A применительно к умножению. Для некоторых целых q и r имеем $a = b + qm$ и $x = y + rm$, поэтому $ax = by + (br + yq + qrm)m$.

18. Для некоторого целого k имеем $a - b = kr$, а также $kr \equiv 0$ (по модулю s). Тогда согласно свойству B $k \equiv 0$ (по модулю s), поэтому $a - b = qsr$ для некоторого целого q.

20. Обе части сравнения умножаем на a'.

21. Согласно ранее рассмотренному упражнению существует по меньшей мере одно такое представление Если предположить, что существует два представления, $n = p_1 \dots p_k = q_1 \dots q_m$, то получаем, что $q_1 \dots q_m \equiv 0$ (по модулю p_1). Поэтому, если ни одно из чисел q_i не равно p_1, то, согласно свойству B их все можно сократить и получить в результате $1 \equiv 0$ (по модулю p_1). Но это невозможно, так как p_1 не равно 1. Поэтому некоторое q_j равно p_1 и $n/p_1 = p_2 \dots p_k = q_1 \dots q_{j-1} q_{j+1} \dots q_m$. Теперь, если n — простое число, теорема доказана; в противном случае можно доказать по индукции, что эти два разложения числа n/p_1 на простые множители одинаковы.

22. Пусть $m = ax$, где $a > 1$ и $x > 0$. Тогда $ax \equiv 0$, но $x \not\equiv 0$ (по модулю m).

24. Свойство A всегда справедливо для операций сложения и вычитания; свойство C справедливо всегда.

26. Если b не кратно p, то $b^2 - 1$ кратно, поэтому один из сомножителей должен делиться на p.

27. Произвольное число взаимно просто с p^e тогда и только тогда, когда оно не кратно p. Поэтому, сосчитав числа, не кратные p, получим $\varphi(p^e) = p^e - p^{e-1}$.

28. Если a и b взаимно просты с m, то и $ab \bmod m$ взаимно просто с m, так как любое простое число, делящее $ab \bmod m$ и m, должно также делить a или b. А теперь пусть числа $x_1, \dots, x_{\varphi(m)}$, меньшие m, взаимно просты с m; заметим, что $ax_1 \bmod m, \dots, ax_{\varphi(m)} \bmod m$ — это те же самые числа, но взятые в ином порядке, и т. д.

29. Докажем (b). Если $r \perp s$ и k^2 делит rs, то найдется такое простое p, что p^2 делит rs. Поэтому p делит, предположим, r и не делит s; следовательно, p^2 делит r. Таким образом, $f(rs) = 0$ тогда и только тогда, когда $f(r) = 0$ или $f(s) = 0$.

30. Предположим, $r \perp s$. Идея доказательства такова: показать, что $\varphi(rs)$ чисел, взаимно простых с rs, — это в точности $\varphi(r)\varphi(s)$ различных чисел $(sx_i + ry_j) \bmod (rs)$, где $x_1, \ldots, x_{\varphi(r)}$ и $y_1, \ldots, y_{\varphi(s)}$ — числа, взаимно простые с r и s соответственно.

Так как φ является мультипликативной функцией, то $\varphi(10^6) = \varphi(2^6)\varphi(5^6) = (2^6 - 2^5)(5^6 - 5^5) = 400000$. А в общем случае, когда $n = p_1^{e_1} \ldots p_r^{e_r}$, имеем

$$\varphi(n) = (p_1^{e_1} - p_1^{e_1-1}) \ldots (p_r^{e_r} - p_r^{e_r-1}) = n \prod_{p \backslash n,\, p-\text{простое}} (1 - 1/p).$$

(Другой вариант доказательства приводится в упр. 1.3.3–27.)

31. Воспользуйтесь тем фактом, что делители rs можно единственным образом записать в виде cd, где c делит r, а d делит s. Аналогично, если $f(n) \geq 0$, можно показать, что функция $\max_{d \backslash n} f(d)$ мультипликативна (см. упр. 1.2.3–35).

33. Либо $n + m$, либо $n - m + 1$ четно, поэтому одна из величин в квадратных скобках является целой. Таким образом, нестрогое неравенство из упр. 7 обращается в равенство и на этом основании получаем: (a) n; (b) $n + 1$.

34. b должно быть целым числом ≥ 2. (Положим $x = b$.) Достаточность доказывается, как в упр. 6. Это же условие является необходимым и достаточным для того, чтобы $\lceil \log_b x \rceil = \lceil \log_b \lceil x \rceil \rceil$.

Примечание. Р. Дж. Мак-Элис (R. J. McEliece) указал обобщение этого результата. Пусть f — непрерывная, строго возрастающая функция, определенная на интервале A. Предположим, что если x принадлежит A, то и значения функций $\lfloor x \rfloor$ и $\lceil x \rceil$ принадлежат A. Тогда утверждения "$\lfloor f(x) \rfloor = \lfloor f(\lfloor x \rfloor) \rfloor$ для всех x из A" и "$\lceil f(x) \rceil = \lceil f(\lceil x \rceil) \rceil$ для всех x из A" равносильны и выполняются тогда и только тогда, когда для всех x из A удовлетворяется следующее условие: "если значением функции $f(x)$ является целое число, то x тоже целое число". Очевидно, что данное условие является необходимым, так как если значением $f(x)$ является целое число, равное $\lfloor f(\lfloor x \rfloor) \rfloor$ или $\lceil f(\lceil x \rceil) \rceil$, то x должно равняться $\lfloor x \rfloor$ или $\lceil x \rceil$. И обратно, если, например, $\lfloor f(\lfloor x \rfloor) \rfloor < \lfloor f(x) \rfloor$, то в силу непрерывности функции $f(x)$ существует такое y, $\lfloor x \rfloor < y \leq x$, для которого значение $f(y)$ является целым, но y не может быть целым числом.

35. $\dfrac{x+m}{n} - 1 = \dfrac{x+m}{n} - \dfrac{1}{n} - \dfrac{n-1}{n} < \dfrac{\lfloor x \rfloor + m}{n} - \dfrac{n-1}{n} \leq \left\lfloor \dfrac{\lfloor x \rfloor + m}{n} \right\rfloor \leq \dfrac{x+m}{n}$. А теперь воспользуйтесь упр. 3. Применив упр. 4, можно получить аналогичный результат для функции "потолок". Оба тождества являются частными случаями теоремы Мак-Элиса из упр. 34.

36. Предположим сначала, что $n = 2t$. Тогда

$$\sum_{k=1}^{n} \left\lfloor \frac{k}{2} \right\rfloor = \sum_{k=1}^{n} \left\lfloor \frac{n+1-k}{2} \right\rfloor;$$

следовательно, с помощью результатов упр. 33 получаем

$$\sum_{k=1}^{n} \left\lfloor \frac{k}{2} \right\rfloor = \frac{1}{2} \sum_{k=1}^{n} \left(\left\lfloor \frac{k}{2} \right\rfloor + \left\lfloor \frac{n+1-k}{2} \right\rfloor \right) = \frac{1}{2} \sum_{k=1}^{n} \left\lfloor \frac{2t+1}{2} \right\rfloor = t^2 = \frac{n^2}{4}.$$

И, если $n = 2t + 1$, имеем $t^2 + \lfloor n/2 \rfloor = t^2 + t = n^2/4 - 1/4$. Аналогично для второй суммы получаем $\lceil n(n+2)/4 \rceil$.

37. $\sum_{0\le k<n} \frac{mk+x}{n} = \frac{m(n-1)}{2} + x$. Обозначим через $\{y\}$ величину $y \bmod 1$. Чтобы получить требуемую сумму, мы должны вычесть из предыдущего равенства

$$S = \sum_{0\le k<n} \left\{\frac{mk+x}{n}\right\}.$$

Величина S состоит из d экземпляров одной и той же суммы, так как если $t = n/d$, то имеем

$$\left\{\frac{mk+x}{n}\right\} = \left\{\frac{m(k+t)+x}{n}\right\}.$$

Положим $u = m/d$; тогда

$$\sum_{0\le k<t} \left\{\frac{mk+x}{n}\right\} = \sum_{0\le k<t} \left\{\frac{x}{n} + \frac{uk}{t}\right\}.$$

Поскольку $t \perp u$, последняя сумма равна

$$\left\{\frac{x \bmod d}{n}\right\} + \left\{\frac{x \bmod d}{n} + \frac{1}{t}\right\} + \cdots + \left\{\frac{x \bmod d}{n} + \frac{t-1}{t}\right\}.$$

И наконец, так как $(x \bmod d)/n < 1/t$, то, опустив скобки в этой сумме, получим

$$S = d\left(\frac{t(x \bmod d)}{n} + \frac{t-1}{2}\right).$$

Применяя упр. 4, получим тождество

$$\sum_{0\le k<n} \left\lceil \frac{mk+x}{n} \right\rceil = \frac{(m+1)(n-1)}{2} - \frac{d-1}{2} + d\lceil x/d \rceil.$$

Если суммирование распространить на область $0 \le k \le n$, формула станет симметричной относительно m и n. (Эту симметрию можно объяснить, нарисовав график функции, стоящей под знаком суммы, как функции от k, а затем отразив его относительно прямой $y = x$.)

38. Обе стороны равенства увеличиваются на $\lceil y \rceil$, когда x увеличивается на 1, поэтому можно предположить, что $0 \le x < 1$. При $x = 0$ обе части обращаются в нуль. Когда x возрастает, при каждом переходе значения $1 - k/y$ при $y > k \ge 0$ обе части увеличиваются на 1. [*Crelle* **136** (1909), 42; случай $y = n$ был рассмотрен Ш. Эрмитом (C. Hermite), *Acta Math.* **5** (1884), 315.]

39. Докажем п. (f). Рассмотрим более общее тождество $\prod_{0\le k<n} 2\sin\pi(x+k/n) = 2\sin\pi nx$. Поскольку $2\sin\theta = (e^{\imath\theta} - e^{-\imath\theta})/i = (1 - e^{-2\imath\theta})e^{\imath\theta - \imath\pi/2}$, данное тождество является следствием следующих двух формул:

$$\prod_{0\le k<n} (1 - e^{-2\pi(x+\imath k/n)}) = 1 - e^{-2\pi nx} \quad \text{и} \quad \prod_{0\le k<n} e^{\pi(x-(1/2)+(k/n))} = e^{\pi(nx-1/2)}.$$

Вторая формула верна, так как функция $x - \frac{1}{2}$ репликативна, а первая верна, так как в разложении многочлена на множители $z^n - \alpha^n = (z-\alpha)(z-\omega\alpha)\ldots(z-\omega^{n-1}\alpha)$, где $\omega = e^{-2\pi\imath/n}$, можно положить $z = 1$.

40. (Замечание Н. Г. де Брейна (N. G. de Bruijn).) Если f репликативна, то $f(nx+1)-f(nx)=f(x+1)-f(x)$ для всех $n>0$. Поэтому, если f непрерывна, то $f(x+1)-f(x)=c$ для всех x и, следовательно, $g(x)=f(x)-c\lfloor x\rfloor$ репликативна и периодична. Далее,

$$\int_0^1 e^{2\pi i n x} g(x)\,dx = \frac{1}{n}\int_0^1 e^{2\pi i y} g(y)\,dy;$$

и разложение в ряд Фурье показывает, что $g(x)=(x-\frac{1}{2})a$ при $0<x<1$. Отсюда следует, что $f(x)=(x-\frac{1}{2})a$. Если говорить в общем, это означает, что любая репликативная локально интегрируемая по Риману функция почти всюду равна $(x-\frac{1}{2})a+b\max(\lfloor x\rfloor,0)+c\min(\lfloor x\rfloor,0)$. Дополнительную информацию по данной теме можно найти в работах L. J. Mordell, *J. London Math. Soc.* **33** (1958), 371-375; M. F. Yoder, *Æquationes Mathematicæ* **13** (1975), 251–261.

41. Нам нужно, чтобы $a_n=k$ при $\frac{1}{2}k(k-1)<n\le\frac{1}{2}k(k+1)$. Так как n — целое число, данное неравенство эквивалентно следующему.

$$\frac{k(k-1)}{2}+\frac{1}{8}<n<\frac{k(k+1)}{2}+\frac{1}{8},$$

т. е. $k-\frac{1}{2}<\sqrt{2n}<k+\frac{1}{2}$. Поэтому $a_n=\lfloor\sqrt{2n}+\frac{1}{2}\rfloor$, т. е. ближайшему к $\sqrt{2n}$ целому числу. Можно привести и другие правильные ответы: $\lceil\sqrt{2n}-\frac{1}{2}\rceil$, $\lceil(\sqrt{8n+1}-1)/2\rceil$, $\lfloor(\sqrt{8n-7}+1)/2\rfloor$ и т. д.

42. (a) См. упр. 1.2.7–10. (b) Заданная сумма равна $n\lfloor\log_b n\rfloor-S$, где

$$S=\sum_{\substack{1\le k<n\\ k+1\text{ есть степень b}}} k=\sum_{1\le t\le\log_b n}(b^t-1)=(b^{\lfloor\log_b n\rfloor+1}-b)/(b-1)-\lfloor\log_b n\rfloor.$$

43. $\lfloor\sqrt{n}\rfloor\,(n-\frac{1}{6}(2\lfloor\sqrt{n}\rfloor+5)(\lfloor\sqrt{n}\rfloor-1))$.

44. Для отрицательных n эта сумма равна $n+1$.

45. $\lfloor mj/n\rfloor=r$ тогда и только тогда, когда $\left\lceil\frac{rn}{m}\right\rceil\le j<\left\lceil\frac{(r+1)n}{m}\right\rceil$, следовательно, заданная сумма равна

$$\sum_{0\le r<m} f(r)\left(\left\lceil\frac{(r+1)n}{m}\right\rceil-\left\lceil\frac{rn}{m}\right\rceil\right).$$

Требуемый результат получаем путем преобразования последней суммы и группирования членов при $\lceil rn/m\rceil$. Вторую формулу немедленно получаем в результате подстановки:

$$f(x)=\binom{x+1}{k}.$$

46. $\sum_{0\le j<\alpha n} f(\lfloor mj/n\rfloor)=\sum_{0\le r<\alpha m}\lceil rn/m\rceil(f(r-1)-f(r))+\lceil\alpha n\rceil f(\lceil\alpha m\rceil-1)$.

47. (a) Числа $2,\ 4,\ \ldots,\ p-1$ — это четные вычеты (по модулю p); поскольку $2kq=p\lfloor 2kq/p\rfloor+(2kq)\bmod p$, число $(-1)^{\lfloor 2kq/p\rfloor}((2kq)\bmod p)$ — четный вычет или четный вычет минус p, причем очевидно, что каждый четный вычет встречается только один раз. Поэтому $(-1)^\sigma q^{(p-1)/2}2\cdot 4\ldots(p-1)\equiv 2\cdot 4\ \ldots(p-1)$. (b) Пусть $q=2$. Если $p=4n+1$, то $\sigma=n$; если $p=4n+3$, то $\sigma=n+1$. Поэтому $\left(\frac{2}{p}\right)=(1,-1,-1,1)$, если $p\bmod 8=(1,3,5,7)$ соответственно. (c) Для $k<p/4$ имеем

$$\lfloor(p-1-2k)q/p\rfloor=q-\lceil(2k+1)q/p\rceil=q-1-\lfloor(2k+1)q/p\rfloor\equiv\lfloor(2k+1)q/p\rfloor\ (\text{по модулю }2).$$

Поэтому мы можем заменить последние члены $\lfloor(p-1)q/p\rfloor$, $\lfloor(p-3)q/p\rfloor,\ldots$ на $\lfloor q/p\rfloor$, $\lfloor 3q/p\rfloor$ и т. д. (d) $\sum_{0\le k<p/2}\lfloor kq/p\rfloor+\sum_{0\le r<q/2}\lceil rp/q\rceil=\lceil p/2\rceil(\lceil q/2\rceil-1)=(p+1)(q-1)/4$. Кроме того, $\sum_{0\le r<q/2}\lceil rp/q\rceil=\sum_{0\le r<q/2}\lfloor rp/q\rfloor+(q-1)/2$. Идея этого доказательства восходит

к Г. Айзенштайну (G. Eisenstein), *Crelle* **28** (1844), 246–248; в этом же томе журнала Айзенштайн дает несколько различных доказательств данного, а также других законов взаимности.

48. (a) Очевидно, что при $n < 0$ это тождество не всегда верно; но легко проверить, что оно справедливо при $n > 0$. (b) $\lfloor(n+2-\lfloor n/25\rfloor)/3\rfloor = \lceil(n-\lfloor n/25\rfloor)/3\rceil = \lceil(n+\lceil -n/25\rceil)/3\rceil = \lceil\lceil 24n/25\rceil/3\rceil = \lceil 8n/25\rceil = \lfloor(8n+24)/25\rfloor$. Предпоследнее равенство доказано в упр. 35.

49. Так как $f(0) = f(f(0)) = f(f(0)+0) = f(0)+f(0)$, то $f(n) = n$ для всех целых n. Если $f(\frac{1}{2}) = k \le 0$, то $k = f(\frac{1}{1-2k}f(\frac{1}{2}-k)) = f(\frac{1}{1-2k}(f(\frac{1}{2})-k)) = f(0) = 0$. И если $f(\frac{1}{n-1}) = 0$, то $f(\frac{1}{n}) = f(\frac{1}{n}f(1+\frac{1}{n-1})) = f(\frac{1}{n-1}) = 0$ и для $1 \le m < n$ индукцией по m можно доказать, что $f(\frac{m}{n}) = f(\frac{1}{a}f(\frac{am}{n})) = f(\frac{1}{a}) = 0$, где $a = \lceil n/m\rceil$. Поэтому, так как $f(\frac{1}{2}) \le 0$, то $f(x) = \lfloor x\rfloor$ для всех рациональных x. С другой стороны, если $f(\frac{1}{2}) > 0$, то функция $g(x) = -f(-x)$ удовлетворяет условиям (i) и (ii) и $g(\frac{1}{2}) = 1 - f(\frac{1}{2}) \le 0$; поэтому $f(x) = -g(-x) = -\lfloor -x\rfloor = \lceil x\rceil$ для всех рациональных x. [P. Eisele, K. P. Hadeler, *AMM* **97** (1990), 475–477.]

Но отсюда не следует, что $f(x) = \lfloor x\rfloor$ или $\lceil x\rceil$ для всех *действительных* значений x. Если, например, $h(x)$ — это произвольная функция, удовлетворяющая условиям $h(1) = 1$ и $h(x+y) = h(x)+h(y)$ для всех действительных x и y, то функция $f(x) = \lfloor h(x)\rfloor$ удовлетворяет условиям (i) и (ii). Но при $0 < x < 1$ функция $h(x)$ может быть неограниченной и достаточно непредсказуемой [G. Hamel, *Math. Annalen* **60** (1905), 459–462].

РАЗДЕЛ 1.2.5

1. 52!. Для тех, кого это интересует, данное число равно 806 58175 17094 38785 71660 63685 64037 66975 28950 54408 83277 82400 00000 00000 (!).

2. $p_{nk} = p_{n(k-1)}(n-k+1)$. После размещения первых $n-1$ объектов для последнего объекта остается только одна возможность.

3. 53124, 35124, 31524, 31254, 31245; 42351, 41352, 41253, 31254, 31245.

4. Всего в этом числе 2 568 цифр. В старшем разряде стоит цифра 4 (так как $\log_{10} 4 = 2\log_{10} 2 \approx .602$). В младшем разряде стоит цифра нуль, и из (8) следует, что в младших 249 разрядах стоят все нули. Точное значение 1000! было вычислено Г. С. Улером (H. S. Uhler) с помощью арифмометра и многолетнего терпения и опубликовано в *Scripta Mathematica* **21** (1955), 266–267. Это число начинается цифрами 402 38726 00770 .... Последнюю точку в этой истории поставил Джон В. Ренч (мл.) (John W. Wrench, Jr.), когда перемножил числа 750! и $\prod_{k=751}^{1000} k$ на компьютере UNIVAC I "за рекордно короткое время — 2½ минуты". В наше время на любом персональном компьютере можно легко вычислить 1000! в течение доли секунды и убедиться в том, что полученное Улером значение на 100% верно.

5. $(39902)(97/96) \approx 416 + 39902 = 40318$.

6. $2^{18}\cdot 3^8\cdot 5^4\cdot 7^2\cdot 11\cdot 13\cdot 17\cdot 19$.

8. Этот предел равен $\lim_{m\to\infty} m^n m!/((n+m)!/n!) = n!\lim_{m\to\infty} m^n/((m+1)\dots(m+n)) = n!$, так как $m/(m+k) \to 1$.

9. $\sqrt{\pi}$ и $-2\sqrt{\pi}$. (Использовано упр. 10.)

10. Да, за исключением случаев, когда x — отрицательное целое или равно нулю. Действительно,

$$\Gamma(x+1) = x\lim_{m\to\infty}\frac{m^x m!}{x(x+1)\dots(x+m)}\left(\frac{m}{x+m+1}\right).$$

11, 12. $\mu = (a_k p^{k-1}+\dots+a_1)+(a_k p^{k-2}+\dots+a_2)+\dots+a_k$
$= a_k(p^{k-1}+\dots+p+1)+\dots+a_1 = (a_k(p^k-1)+\dots+a_0(p^0-1))/(p-1)$
$= (n-a_k-\dots-a_1-a_0)/(p-1)$.

13. Для каждого n, $1 \le n < p$, определите n', как в упр. 1.2 4–19. Согласно свойству 1.2.4D существует ровно одно такое n', и $(n')' = n$. Поэтому мы можем разбить числа на пары при условии, что $n' \ne n$. Если $n' = n$, то $n^2 \equiv 1$ (по модулю p). Следовательно, как и в упр. 1.2.4–26, $n = 1$ или $n = p - 1$. Поэтому $(p-1)! \equiv 1 \cdot 1 \ldots 1 \cdot (-1)$, так как 1 и $p-1$ — это единственные элементы, не имеющие пары.

14. Среди чисел $\{1, 2, \ldots, n\}$, *не* кратных p, существует $\lfloor n/p \rfloor$ полных наборов из $p-1$ последовательных элементов, произведение которых сравнимо с -1 (по модулю p) (по теореме Вильсона). Помимо этого, остается еще a_0 элементов, произведение которых сравнимо с $a_0!$ (по модулю p); поэтому вклад от сомножителей, не кратных p, составляет $(-1)^{\lfloor n/p \rfloor} a_0!$. Вклад от сомножителей, которые *кратны* p, является таким же, как вклад в $\lfloor n/p \rfloor !$. Повторяя эти рассуждения, получим искомую формулу.

15. $(n!)^3$. В формуле содержится $n!$ членов. В каждом члене присутствует по одному элементу из каждой строки и каждого столбца, поэтому его значение равно $(n!)^2$.

16. Члены суммы не стремятся к нулю, так как коэффициенты стремятся к $1/e$.

17. Запишите гамма-функции в виде пределов по формуле (15).

18. $$\prod_{n \ge 1} \frac{n}{n-\frac{1}{2}} \frac{n}{n+\frac{1}{2}} = \frac{\Gamma(\frac{1}{2})\Gamma(\frac{3}{2})}{\Gamma(1)\Gamma(1)} = 2\Gamma(\tfrac{3}{2})^2.$$

[Принадлежащее Валлису эвристическое "доказательство" можно найти в книге D. J. Struik *Source Book in Mathematics* (Harvard University Press, 1969), 244–253.]

19. Сделаем замену переменных $t = mt$, проинтегрируем по частям и проведем доказательство по индукции.

20. [Для полноты картины докажем неравенство, сформулированное в условии задачи. Начнем с легко проверяемого неравенства $1 + x \le e^x$. Положим $x = \pm t/n$ и возведем обе части неравенства в n-ю степень; в результате получим $(1 \pm t/n)^n \le e^{\pm t}$. Следовательно, $e^{-t} \ge (1-t/n)^n = e^{-t}(1-t/n)^n e^t \ge e^{-t}(1-t/n)^n(1+t/n)^n = e^{-t}(1-t^2/n^2)^n \ge e^{-t}(1-t^2/n)$ (см. упр. 1.2.1–9).]

Теперь отнимем от заданного интеграла $\Gamma_m(x)$ и получим

$$\int_m^\infty e^{-t} t^{x-1}\, dt + \int_0^m \left(e^{-t} - \left(1 - \frac{t}{m}\right)^m\right) t^{x-1}\, dt.$$

При $m \to \infty$ первый из этих интегралов стремится к нулю, так как для больших t имеем $t^{x-1} < e^{t/2}$, а второй интеграл меньше, чем

$$\frac{1}{m}\int_0^m t^{x+1} e^{-t}\, dt < \frac{1}{m}\int_0^\infty t^{x+1} e^{-t}\, dt \to 0.$$

21. Если обозначить соответствующий коэффициент через $c(n, j, k_1, k_2, \ldots)$, то после дифференцирования находим

$$c(n+1, j, k_1, \ldots) = c(n, j-1, k_1-1, k_2, \ldots) + (k_1+1)c(n, j, k_1+1, k_2-1, k_3, \ldots) + (k_2+1)c(n, j, k_1, k_2+1, k_3-1, k_4, \ldots) + \cdots.$$

Заметим, что после осуществления шага индукции соотношения $k_1 + k_2 + \cdots = j$ и $k_1 + 2k_2 + \cdots = n$ сохраняются. Коэффициент $n!/(k_1!(1!)^{k_1}k_2!(2!)^{k_2}\ldots)$ можно вынести из каждого члена правой части соотношения для $c(n+1, j, k_1, \ldots)$; в результате останется $k_1 + 2k_2 + 3k_3 + \cdots = n + 1$. (Доказательство удобно проводить в предположении, что существует бесконечно много k_i, хотя очевидно, что $k_{n+1} = k_{n+2} = \cdots = 0$.)

В приведенном доказательстве использовались стандартные методы, но оно не дает удовлетворительного объяснения по поводу того, *почему* формула имеет такой вид и как

она была открыта. Попробуем ответить на этот вопрос с помощью доказательства методами комбинаторики, предложенного Г. С. Воллом (H. S. Wall) [*Bull. Amer. Math. Soc.* **44** (1938), 395–398]. Для удобства введем обозначения $w_j = D_u^j w$, $u_k = D_x^k u$. Тогда $D_x(w_j) = w_{j+1}u_1$ и $D_x(u_k) = u_{k+1}$. Используя эти два соотношения и правило дифференцирования произведения, получим

$$D_x^1 w = w_1 u_1$$
$$D_x^2 w = (w_2 u_1 u_1 + w_1 u_2)$$
$$D_x^3 w = ((w_3 u_1 u_1 u_1 + w_2 u_2 u_1 + w_2 u_1 u_2) + (w_2 u_1 u_2 + w_1 u_3)) \text{ и т. д.}$$

По аналогии можно построить соответствующую таблицу для разбиений множеств:

$$\mathcal{D}^1 = \{1\}$$
$$\mathcal{D}^2 = (\{2\}\{1\} + \{2,1\})$$
$$\mathcal{D}^3 = ((\{3\}\{2\}\{1\} + \{3,2\}\{1\} + \{2\}\{3,1\}) + (\{3\}\{2,1\} + \{3,2,1\})) \text{ и т. д.}$$

Если $a_1 a_2 \ldots a_j$ — разбиение множества $\{1, 2, \ldots, n-1\}$, то формально определим

$$\mathcal{D}a_1 a_2 \ldots a_j = \{n\}a_1 a_2 \ldots a_j + (a_1 \cup \{n\})a_2 \ldots a_j + a_1(a_2 \cup \{n\}) \ldots a_j + \cdots + a_1 a_2 \ldots (a_j \cup \{n\}).$$

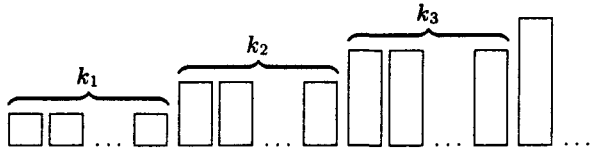
Это правило — точная копия правила дифференцирования

$$D_x(w_j u_{r_1} u_{r_2} \ldots u_{r_j}) = w_{j+1} u_1 u_{r_1} u_{r_2} \ldots u_{r_j} + w_j u_{r_1+1} u_{r_2} \ldots u_{r_j} + w_j u_{r_1} u_{r_2+1} \ldots u_{r_j} + \cdots + w_j u_{r_1} u_{r_2} \ldots u_{r_j+1},$$

если члену $w_j u_{r_1} u_{r_2} \ldots u_{r_j}$ поставить в соответствие разбиение $a_1 a_2 \ldots a_j$ с r_t элементами в a_t, $1 \le t \le j$. Поэтому существует естественное отображение $\mathcal{D}^n$ в $D_x^n w$ и легко видеть, что $\mathcal{D}^n$ содержит каждое разбиение множества $\{1, 2, \ldots, n\}$ ровно один раз (см. упр. 1.2.6–64).

Таким образом, если сгруппировать одинаковые члены в $D_x^n w$, то получим сумму членов вида $c(k_1, k_2, \ldots) w_j u_1^{k_1} u_2^{k_2} \ldots$, где $j = k_1 + k_2 + \cdots$ и $n = k_1 + 2k_2 + \cdots$, а $c(k_1, k_2, \ldots)$ — это *число разбиений множества* $\{1, 2, \ldots, n\}$ *на* j *подмножеств, таких, что существует* k_t *подмножеств, состоящих из* t *элементов*.

Осталось подсчитать эти разбиения. Рассмотрим множества, состоящие из k_t ящиков емкостью t.



Число способов помещения n различных элементов в ячейки — это полиномиальный коэффициент

$$\binom{n}{1,1,\ldots,1,2,2,\ldots,2,3,3,\ldots,3,4,\ldots} = \frac{n!}{1!^{k_1} 2!^{k_2} 3!^{k_3} \ldots}.$$

Чтобы получить $c(k_1, k_2, k_3, \ldots)$, нужно разделить это выражение на $k_1!\, k_2!\, k_3! \ldots$, так как ячейки в каждой группе k_t не отличаются одна от другой и их можно переставить $k_t!$ способами, не оказывая никакого влияния на разбиение множеств.

Оригинальное доказательство Арбогаста [*Du Calcul des Dérivations* (Strasbourg, 1800), §52] основывалось на том факте, что $D_x^k u/k!$ — это коэффициент при z^k в выражении

для $u(x+z)$, а $D_u^j w/j!$ — коэффициент при y^j в выражении для $w(u+y)$. Отсюда следует, что коэффициент при z^n в формуле для $w(u(x+z))$ равен

$$\frac{D_x^n w}{n!} = \sum_{j=0}^{n} \frac{D_u^j w}{j!} \sum_{\substack{k_1+k_2+\ \ +k_n=j \\ k_1+2k_2+\ \ +nk_n=n \\ k_1,k_2,\ \ .,k_n\geq 0}}^{k} \frac{j!}{k_1!\,k_2!\ldots k_n!} \left(\frac{D_x^1 u}{1!}\right)^{k_1} \left(\frac{D_x^2 u}{2!}\right)^{k_2} \ldots \left(\frac{D_x^n u}{n!}\right)^{k_n}.$$

Формула Арбогаста долгие годы оставалась забытой, а затем была вновь независимо открыта Ф. Фаа ди Бруно (F. Faà di Bruno) [*Quarterly J. Math.* **1** (1857), 359–360], который заметил, что производную можно также представить в виде определителя

$$D_x^n = \det \begin{pmatrix} \binom{n-1}{0}u_1 & \binom{n-1}{1}u_2 & \binom{n-1}{2}u_3 & \ldots & \binom{n-1}{n-2}u_{n-1} & \binom{n-1}{n-1}u_n \\ -1 & \binom{n-2}{0}u_1 & \binom{n-2}{1}u_2 & \ldots & \binom{n-2}{n-3}u_{n-2} & \binom{n-2}{n-2}u_{n-1} \\ 0 & -1 & \binom{n-3}{0}u_1 & \ldots & \binom{n-3}{n-4}u_{n-3} & \binom{n-3}{n-3}u_{n-2} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \ldots & -1 & \binom{0}{0}u_1 \end{pmatrix},$$

где $u_j = (D_x^j u)\,D_u$. Обе части этого соотношения представляют собой дифференциальные операторы, которые нужно применить к w. Обобщение формулы Арбогаста для функции многих переменных, а также список ссылок на другие работы по данной теме можно найти в статье И. Дж. Гуда (I. J. Good) *Annals of Mathematical Statistics* **32** (1961), 540–541.

22. Выдвигаем следующую гипотезу: $\lim_{n\to\infty}(n+x)!/(n!\,n^x) = 1$. Она справедлива для целых x. Например, если x положительно, то выражение под знаком предела равно

$$(1+1/n)(1+2/n)\ldots(1+x/n)$$

и при n, стремящемся к бесконечности, оно безусловно стремится к единице. Если предположить также, что $x! = x(x-1)!$, то из гипотезы немедленно следует, что

$$1 = \lim_{n\to\infty} \frac{(n+x)!}{n!\,n^x} = x! \lim_{n\to\infty} \frac{(x+1)\ldots(x+n)}{n!\,n^x},$$

а это эквивалентно определению, данному в тексте.

23. Из (13) и (15) следует, что $z\,(-z)!\,\Gamma(z) = \lim_{m\to\infty} \prod_{n=1}^{m}(1-z/n)^{-1}(1+z/n)^{-1}$.

24. $n^n/n! = \prod_{k=1}^{n-1}(k+1)^k/k^k \leq \prod_{k=1}^{n-1} e$; $n!/n^{n+1} = \prod_{k=1}^{n-1} k^{k+1}/(k+1)^{k+1} \leq \prod_{k=1}^{n-1} e^{-1}$.

25. $x^{\underline{m+n}} = x^{\underline{m}}(x-m)^{\underline{n}}$; $x^{\overline{m+n}} = x^{\overline{m}}(x-m)^{\overline{n}}$. Из (21) следует, что эти правила справедливы также для нецелых m и n.

РАЗДЕЛ 1.2.6

1. n, так как каждое сочетание получается отбрасыванием одного элемента.

2. 1. Существует ровно один способ не выбрать ничего из пустого множества.

3. $\binom{52}{13}$. Это число равно 635 013 559 600.

4. $2^4 \cdot 5^2 \cdot 7^2 \cdot 17 \cdot 23 \cdot 41 \cdot 43 \cdot 47$.

5. $(10+1)^4 = 10000 + 4(1000) + 6(100) + 4(10) + 1$.

6.
$r=-3$: 1 −3 6 −10 15 −21 28 −36 ...
$r=-2$: 1 −2 3 −4 5 −6 7 −8 ...
$r=-1$: 1 −1 1 −1 1 −1 1 −1 ...

7. При $\lfloor n/2 \rfloor$ либо при $\lceil n/2 \rceil$. Из (3) следует, что при меньших значениях биномиальный коэффициент строго возрастает, а при больших убывает до нуля.

8. Если обозначить ненулевые элементы в строке a_1, a_2, $\ldots$, a_n, то они обладают свойством $a_k = a_n - k$.

9. Единице, если n положительно или равно нулю; нулю, если n отрицательно.

10. (a), (b) и (f) непосредственно следуют из (e); (c) и (d) следуют из (a), (b) и (9). Поэтому остается доказать (e). Рассмотрим $\binom{n}{k}$ как дробь, которая задается соотношением (3), где и в числителе, и в знаменателе находится произведение сомножителей. Произведение первых $k \bmod p$ сомножителей в знаменателе не делится на p; очевидно, что эти члены в числителе и знаменателе сравнимы с соответствующими членами биномиального коэффициента

$$\binom{n \bmod p}{k \bmod p},$$

которые отличаются на величины, кратные p. (Если рассматривать величины, не кратные p, то можно брать по модулю p и числитель, и знаменатель, так как если $a \equiv c$, $b \equiv d$ и a/b, c/d — целые, то $a/b \equiv c/d$) Остается $k - k \bmod p$ сомножителей, каждый из которых входит в $\lfloor k/p \rfloor$ групп из p последовательных величин. В каждой группе содержится ровно одна величина, кратная p; остальные $p-1$ сомножителей группы сравнимы (по модулю p) с $(p-1)!$, поэтому они сокращаются в числителе и знаменателе. Остается рассмотреть в числителе и знаменателе $\lfloor k/p \rfloor$ величин, кратных p. Разделив каждую из них на p, получим биномиальный коэффициент

$$\binom{\lfloor (n - k \bmod p)/p \rfloor}{\lfloor k/p \rfloor}.$$

Если $k \bmod p \le n \bmod p$, то этот биномиальный коэффициент равен

$$\binom{\lfloor n/p \rfloor}{\lfloor k/p \rfloor},$$

как и требуется. Если $k \bmod p > n \bmod p$, то другой множитель $\binom{n \bmod p}{k \bmod p}$ равен нулю. Таким образом, наша формула справедлива во всех случаях. [*American J. Math.* **1** (1878), 229–230; см также N. J. Fine, *AMM* **54** (1947), 589–592]

11. Если $a = a_r p^r + \cdots + a_0$, $b = b_r p^r + \cdots + b_0$ и $a + b = c_r p^r + \cdots + c_0$, то значение n (согласно упр. 1.2.5–12 и соотношению (5)) равно

$$(a_0 + \cdots + a_r + b_0 + \cdots + b_r - c_0 - \cdots - c_r)/(p-1).$$

В результате одного переноса c_j уменьшается на p, а c_{j+1} увеличивается на 1, что дает по этой формуле суммарное изменение на $+1$. [Аналогичные утверждения справедливы для q-номиальных и фибономиальных (Fibonomial) коэффициентов; см работу Knuth, Wilf, *Crelle* **396** (1989), 212–219.]

12. Согласно любому из двух предыдущих упражнений n должно быть на единицу меньше степени 2. А если обобщить, то $\binom{n}{k}$ не делится на простое число p, $0 \le k \le n$, тогда и только тогда, когда $n = ap^m - 1$, $1 \le a < p$, $m \ge 0$

14.
$$24\binom{n+1}{5} + 36\binom{n+1}{4} + 14\binom{n+1}{3} + \binom{n+1}{2}$$
$$= \frac{n^5}{5} + \frac{n^4}{2} + \frac{n^3}{3} - \frac{n}{30} = \frac{n(n+1)(n+\frac{1}{2})(3n^2+3n-1)}{15}.$$

15. Проведите доказательство по индукции и воспользуйтесь соотношением (9).

17. Мы можем предположить, что r и s — положительные целые числа. Кроме того, для всех x

$$\sum_n \binom{r+s}{n} x^n = (1+x)^{r+s} = \sum_k \binom{r}{k} x^k \sum_m \binom{s}{m} x^m$$

$$= \sum_k \binom{r}{k} x^k \sum_n \binom{s}{n-k} x^{n-k} = \sum_n \left(\sum_k \binom{r}{k} \binom{s}{n-k} \right) x^n,$$

поэтому коэффициенты при x^n должны быть равны.

21. Левая часть равенства — это многочлен степени n, а правая — многочлен степени $m+n+1$. Они совпадают в $n+1$ точке. Но этого недостаточно для доказательства их тождественности (хотя оказывается, что при $m = 0$ обе части кратны некоторому многочлену; и действительно, при $m = 0$ находим, что равенство *является* тождеством по s, так как оно совпадает с (11)).

22. Предположим, что $n > 0$. k-й член равен

$$\frac{1}{n!}\binom{n}{k} \prod_{0<j<k} (r - tk - j) \prod_{0 \le j < n-k} (n - 1 - r + tk - j)$$

$$= \frac{(-1)^{k-1}}{n!}\binom{n}{k} \prod_{0<j<k} (-r + tk + j) \prod_{k \le j < n} (-r + tk + j),$$

и эти два произведения дают многочлен степени $n-1$ по k. Поэтому согласно (34) сумма по k равна нулю.

24. Докажем индукцией по n. При $n \le 0$ тождество очевидно. Если $n > 0$, докажем индукцией по целому $m \ge 0$, что оно выполняется для $(r, n-r+nt+m, t, n)$ Для этого воспользуемся двумя предыдущими упражнениями и тем, что тождество справедливо для $n-1$. Таким образом, тождество (r, s, t, n) выполняется для бесконечно большого числа значений s; более того, оно выполняется для всех s, так как обе его части являются многочленами по s.

25. Используя критерий отношений* и простые оценки для больших значений k, можно доказать сходимость ряда. (Есть и другой способ: воспользоваться теорией комплексного переменного и показать, что функция является аналитической в окрестности точки $x = 1$.) Имеем

$$1 = \sum_{k,j} (-1)^j \binom{k}{j} \binom{r-jt}{k} \frac{r}{r-jt} w^k = \sum_j (-1)^j \frac{r}{r-jt} \sum_k \binom{k}{j} \binom{r-jt}{k} w^k$$

$$= \sum_j \frac{(-1)^j r}{r-jt} \sum_k \binom{r-jt}{j} \binom{r-jt-j}{k-j} w^k = \sum_j (-1)^j A_j(r,t)(1+w)^{r-jt-j} w^j.$$

Теперь положим $x = 1/(1+w)$, $z = -w/(1+w)^{1+t}$. Это доказательство принадлежит Г. У. Гоулду (H. W. Gould) [*AMM* **63** (1956), 84–91]. См. также более общие формулы в упр. 2.3.4.4–33 и 4.7–22.

26. Можно начать с тождества (35)

$$\sum_j (-1)^j \binom{k}{j} \binom{r-jt}{k} = t^k$$

и продолжать по той же схеме, которая использовалась в упр 25. Другой способ состоит в том, чтобы продифференцировать нашу формулу по z; тогда получим

$$\sum_k k A_k(r,t) z^k = z \frac{d(x^r)}{dz} = \frac{(x^{t+1} - x^t) r x^r}{(t+1)x^{t+1} - t x^t},$$

---

* Критерий д'Аламбера. — *Прим. ред.*

а это уже позволяет вычислить значение суммы

$$\sum_k \Big(1 - \frac{t}{r}k\Big) A_k(r,t) z^k.$$

27. Чтобы получить (26), умножаем ряд для $x^{r+1}/((t+1)x-t)$ на ряд для x^s и получаем ряд для $x^{r+s+1}/((t+1)x-t)$, в котором коэффициенты при z нужно приравнять к соответствующим коэффициентам ряда для $x^{(r+s)+1}/((t+1)x-t)$.

28. Обозначив левую часть через $f(r,s,t,n)$, с помощью тождества

$$\sum_k \binom{r+tk}{k}\binom{s-tk}{n-k}\frac{r}{r+tk} + \sum_k \binom{r+tk}{k}\binom{s-tk}{n-k}\frac{tk}{r+tk} = f(r,s,t,n)$$

находим

$$\binom{r+s}{n} + t f(r-t-1,\, s+t,\, t,\, n-1) = f(r,s,t,n).$$

29. $(-1)^k\binom{n}{k}\Big/n! = (-1)^k/(k!\,(n-k)!) = (-1)^n\Big/\prod_{\substack{0\le j\le n\\ j\ne k}}(k-j).$

30. Применяя (7), (6) и (19), получаем

$$\sum_{k\ge 0}\binom{-m-2k-1}{n-m-k}\binom{2k+1}{k}\frac{(-1)^{n-m}}{2k+1}.$$

Теперь можно применить (26), заменив $(r,s,t,n) = (1, m-2n-1, -2, n-m)$. Тогда получим

$$(-1)^{n-m}\binom{-m}{n-m} = \binom{n-1}{n-m}.$$

Для положительного n этот результат совпадает с формулой, полученной в тексте, но при $n=0$ наш результат справедлив, а $\binom{n-1}{m-1}$ — нет. Данное решение имеет еще одно преимущество: ответ $\binom{n-1}{n-m}$ справедлив для $n \ge 0$ и для *всех* целых m.

31. [Эта сумма была впервые получена в замкнутой форме И. Ф. Пфаффом (J. F. Pfaff), *Nova Acta Acad. Scient. Petr.* **11** (1797), 38–57.] Имеем

$$\begin{aligned}
&\sum_k\sum_j \binom{m-r+s}{k}\binom{n+r-s}{n-k}\binom{r}{m+n-j}\binom{k}{j}\\
&\quad= \sum_j\sum_k \binom{m-r+s}{j}\binom{n+r-s}{n-k}\binom{r}{m+n-j}\binom{m-r+s-j}{k-j}\\
&\quad= \sum_j \binom{m-r+s}{j}\binom{r}{m+n-j}\binom{m+n-j}{n-j}.
\end{aligned}$$

Заменив $\binom{m+n-j}{n-j}$ на $\binom{m+n-j}{m}$ и снова применив (20), получаем

$$\sum_j \binom{m-r+s}{j}\binom{r}{m}\binom{r-m}{n-j} = \binom{r}{m}\binom{s}{n}.$$

32. В формуле (44) замените x значением $-x$.

33, 34. [*Giornale di Mat. Battaglini* **31** (1893), 291–313; **33** (1895), 179–182.] Имеем $x^{\overline{n}} = n!\binom{x+n-1}{n}$. Следовательно, наше соотношение можно преобразовать следующим образом:

$$\binom{x+y+n-1}{n} = \sum_k \binom{x+(1-z)k}{k}\binom{y-1+nz+(n-k)(1-z)}{n-k}\frac{x}{x+(1-z)k},$$

а это частный случай (26). Аналогично $(x+y)^{\underline{n}} = \sum_k \binom{n}{k} x(x-kz-1)^{\underline{k-1}}(y+kz)^{\underline{n-k}}$, что эквивалентно формуле Роте [*Formulæ de Serierum Reversione* (Leipzig, 1793), 18].

35. Например, докажем первую формулу:

$$\sum_k (-1)^{n+1-k} \left(n\begin{bmatrix} n \\ k \end{bmatrix} + \begin{bmatrix} n \\ k-1 \end{bmatrix} \right) x^k = -nx^{\underline{n}} + xx^{\underline{n}} = x^{\underline{n+1}}.$$

36. Из (13), предполагая, что n — неотрицательное целое, для искомых сумм получим значения 2^n и δ_{n0} соответственно.

37. Для $n > 0$ эта сумма равна 2^{n-1}. (Четные и нечетные члены взаимно уничтожаются, поэтому сумма как четных, так и нечетных членов равна половине общей суммы.)

38. Положим $\omega = e^{2\pi\imath/m}$. Тогда

$$\sum_{0\le j<m} (1+\omega^j)^n \omega^{-jk} = \sum_t \sum_{0\le j<m} \binom{n}{t} \omega^{j(t-k)}.$$

Так как

$$\sum_{0\le j<m} \omega^{rj} = m\,[r \equiv 0 (\text{по модулю } m)]$$

(это сумма геометрической прогрессии), сумма справа равна $m \sum_{t \bmod m = k} \binom{n}{t}$. Первоначальная сумма слева равна

$$\sum_{0\le j<m} (\omega^{-j/2} + \omega^{j/2})^n \omega^{j(n/2-k)} = \sum_{0\le j<m} \left(2\cos\frac{j\pi}{m}\right)^n \omega^{j(n/2-k)}$$

Поскольку известно, что эта величина действительна, можем взять действительную часть и получить искомую формулу.

Случаи $m = 3$ и $m = 5$ имеют особые свойства, описанные в *CMath*, упр. 5.75 и 6.57.

39. $n!$; $\delta_{n0} - \delta_{n1}$. (Суммы чисел из строк второго треугольника выглядят уже не так просто; далее (в упр. 64) мы увидим, что $\sum_k \left\{ n \atop k \right\}$ — это число способов разбиения множества из n элементов на непересекающиеся подмножества, т. е. число отношений эквивалентности на множестве $\{1, 2, \ldots, n\}$.)

40. Доказательство (c). Интегрируя по частям, получим

$$\mathrm{B}(x+1, y) = -\frac{t^x(1-t)^y}{y}\Big|_0^1 + \frac{x}{y}\int_0^1 t^{x-1}(1-t)^y\,dt.$$

А теперь применим (b).

41. $m^x \mathrm{B}(x, m+1) \to \Gamma(x)$ при $m \to \infty$. Достаточно рассмотреть только целые значения m (в силу монотонности). Следовательно, $(m+y)^x \mathrm{B}(x, m+y+1) \to \Gamma(x)$ и $\big(m/(m+y)\big)^x \to 1$.

42. $1/((r+1)\mathrm{B}(k+1, r-k+1))$, если $B(x,y)$ определено согласно упр. 41, (b). В общем случае, если z и w — произвольные комплексные числа, определим

$$\binom{z}{w} = \lim_{\zeta\to z}\lim_{\omega\to w} \frac{\zeta!}{\omega!\,(\zeta-\omega)!}, \qquad \text{где } \zeta! = \Gamma(\zeta+1);$$

это значение равно бесконечности, если z — отрицательное целое и w не является целым числом

При таком определении свойство симметрии (6) выполняется для всех комплексных n и k за исключением случаев, когда n — отрицательное целое и k — целое. Соотношения (7),

(9) и (20) справедливы всегда, хотя иногда они могут быть неопределенными, например $0 \quad \infty$ или $\infty + \infty$ Соотношение (17) принимает вид

$$\binom{z}{w} = \frac{\sin \pi(w - z - 1)}{\sin \pi z} \binom{w - z - 1}{w}$$

Можно распространить на область комплексных чисел даже биномиальную теорему (13) и свертку Вандермонда (21) Тогда получим $\sum_k \binom{r}{\alpha+k} z^{\alpha+k} = (1+z)^r$ и $\sum_k \binom{r}{\alpha+k}\binom{s}{\beta-k} = \binom{r+s}{\alpha+\beta}$, эти формулы выполняются для всех комплексных r, s, z, α и β, при которых ряд сходится, если степени комплексных чисел определены соответствующим образом [См L Ramshaw, *Inf Proc Letters* **6** (1977), 223–226]

43. $\int_0^1 dt/(t^{1/2}(1-t)^{1/2}) = 2\int_0^1 du/(1-u^2)^{1/2} = 2 \arcsin u\big|_0^1 = \pi$

45. Для больших r получаем $\dfrac{1}{k\Gamma(k)}\sqrt{\dfrac{r}{r-k}}\,\dfrac{1}{e^k}\,\dfrac{(1-k/r)^k}{(1-k/r)^r} \to \dfrac{1}{\Gamma(k+1)}$

46. $\sqrt{\dfrac{1}{2\pi}\left(\dfrac{1}{x}+\dfrac{1}{y}\right)}\left(1+\dfrac{y}{x}\right)^x\left(1+\dfrac{x}{y}\right)^y$ и $\dbinom{2n}{n} \approx 4^n/\sqrt{\pi n}$

47. При $k \le 0$ каждый элемент тождества равен δ_{k0} и умножается на $(r-k)(r-\frac{1}{2}-k)/(k+1)^2$, когда k заменяется на $k+1$ При $r = -\frac{1}{2}$ отсюда следует, что $\binom{-1/2}{k} = (-1/4)^k\binom{2k}{k}$

48. Это можно доказать по индукции, воспользовавшись тем, что при $n > 0$

$$0 = \sum_k \binom{n}{k}(-1)^k = \sum_k \binom{n}{k}\frac{(-1)^k k}{k+x} + \sum_k \binom{n}{k}\frac{(-1)^k x}{k+x}$$

Другой способ имеем

$$\mathrm{B}(x,\, n+1) = \int_0^1 t^{x-1}(1-t)^n\, dt = \sum_k \binom{n}{k}(-1)^k \int_0^1 t^{x+k-1}\, dt$$

(Фактически наша сумма равна $\mathrm{B}(x,\, n+1)$ и для нецелых n, для которых ряд сходится)

49. $\dbinom{r}{m} = \sum_k \dbinom{r}{k}\dbinom{-r}{m-2k}(-1)^{m+k}$, где m — целое (См упр 17)

50. k-е слагаемое равно $\binom{n}{k}(-1)^{n-k}(x-kz)^{n-1}x$ Примените соотношение (34)

51. Выражение в правой части равно

$$\sum_k \binom{n}{n-k} x(x-kz)^{k-1} \sum_j \binom{n-k}{j}(x+y)^j(-x+kz)^{n-k-j}$$

$$= \sum_j \binom{n}{j}(x+y)^j \sum_k \binom{n-j}{n-j-k} x(x-kz)^{k-1}(-x+kz)^{n-k-j}$$

$$= \sum_{j \le n} \binom{n}{j}(x+y)^j 0^{n-j} = (x+y)^n$$

Точно так же можно вычислить сумму Торелли (упр 34)

Другое красивое доказательство формулы Абеля получается из того, что эта формула легко преобразуется в более симметричное тождество, приведенное в упр 2 3 4 4–29

$$\sum_k \binom{n}{k} x(x+kz)^{k-1} y(y+(n-k)z)^{n-k-1} = (x+y)(x+y+nz)^{n-1}.$$

А. Гурвиц (A. Hurwitz) [*Acta Mathematica* **26** (1902), 199–203] еще больше обобщил теорему Абеля следующим образом:

$$\sum x(x+\epsilon_1 z_1+\cdots+\epsilon_n z_n)^{\epsilon_1+\cdots+\epsilon_n-1}(y-\epsilon_1 z_1-\cdots-\epsilon_n z_n)^{n-\epsilon_1-\cdots-\epsilon_n}=(x+y)^n,$$

где сумма берется по всем 2^n наборам $\epsilon_1,\ldots,\epsilon_n$, независимо принимающим значения 0 или 1. Это тождество по $x, y, z_1, \ldots, z_n$, а формула Абеля — частный случай, когда $z_1 = z_2 = \cdots = z_n$. Формула Гурвица следует из результата упр. 2.3.4.4–30.

52. $\sum_{k\ge 0}(k+1)^{-2} = \pi^2/6$. [М. Л. И. Хаутус (M. L. J. Hautus) заметил, что эта сумма абсолютно сходится для всех комплексных x, y, z, n, когда $z \neq 0$, так как для больших k члены суммы всегда имеют порядок $1/k^2$. На ограниченных областях эта сходимость является равномерной, поэтому можно продифференцировать ряд почленно. Обозначив через $f(x,y,n)$ значение суммы при $z = 1$, находим $(\partial/\partial y)f(x,y,n) = nf(x,y,n-1)$ и $(\partial/\partial x)f(x,y,n) = nf(x-1,y+1,n-1)$. Эти формулы не противоречат тому, что $f(x,y,n) = (x+y)^n$. Но на самом деле последнее равенство, видимо, выполняется редко (если выполняется вообще), если только сумма не является конечной. К тому же производная по z почти всегда является ненулевой.]

53. Для доказательства п. (b) в формуле из п. (a) положите $r = \frac{1}{2}$ и $s = -\frac{1}{2}$.

54. Вставьте знаки "минус" в шахматном порядке, как показано ниже.

$$\begin{pmatrix} 1 & -0 & 0 & -0 \\ -1 & 1 & -0 & 0 \\ 1 & -2 & 1 & -0 \\ -1 & 3 & -3 & 1 \end{pmatrix}$$

Это эквивалентно умножению a_{ij} на $(-1)^{i+j}$. Согласно (33) это и есть искомая обратная матрица.

55. Вставив знаки "минус" в один треугольник (как в предыдущем упражнении), получим обратную матрицу другого треугольника. (Соотношение (47).)

56. 012 013 023 123 014 024 124 034 134 234 015 025 125 035 135 235 045 145 245 345 016. Если зафиксировать c, то пара a, b пробегает комбинации из c чисел по два; если зафиксировать c и b, то a пробегает комбинации из b чисел по одному.

Аналогично можно представить все числа в виде $n = \binom{a}{1}+\binom{b}{2}+\binom{c}{3}+\binom{d}{4}$, где $0 \le a < b < c < d$. Эта последовательность начинается с 0123 0124 0134 0234 1234 0125 0135 0235 .... Комбинаторное представление можно найти по принципу максимализма: сначала выбрать наибольшее возможное d, затем — наибольшее возможное c для $n - \binom{d}{4}$ и т. д. [Другие свойства этого представления обсуждаются в разделе 7.2.1.]

58. Проведем доказательство по индукции, так как

$$\binom{n}{k}_q = \binom{n-1}{k}_q + \binom{n-1}{k-1}_q q^{n-k} = \binom{n-1}{k}_q q^k + \binom{n-1}{k-1}_q.$$

Отсюда получаем, что обобщение соотношения (21) на случай, когда q не равно 1, выглядит следующим образом:

$$\sum_k \binom{r}{k}_q \binom{s}{n-k}_q q^{(r-k)(n-k)} = \sum_k \binom{r}{k}_q \binom{s}{n-k}_q q^{(s-n+k)k} = \binom{r+s}{n}_q.$$

И с помощью тождества $1 - q^t = -q^t(1-q^{-t})$ можно легко обобщить (17):

$$\binom{r}{k}_q = (-1)^k \binom{k-r-1}{k}_q q^{kr-k(k-1)/2}.$$

q-номиальные коэффициенты имеют много различных применений; см., например, раздел 5.1.2 и авторские примечания в *J. Combinatorial Theory* **A10** (1971), 178–180.

Полезные факты. Если n — неотрицательное целое число, то $\binom{n}{k}_q$ — многочлен степени $k(n-k)$ от q с неотрицательными целыми коэффициентами, удовлетворяющий рефлексивному закону:

$$\binom{n}{k}_q = \binom{n}{n-k}_q = q^{k(n-k)}\binom{n}{k}_{q^{-1}}$$

Для $|q|<1$ и $|x|<1$ q-номиальная теорема справедлива для произвольного действительного числа n, если заменить левую часть выражением $\prod_{k\geq 0}((1+q^k x)/(1+q^{n+k}x))$. Благодаря свойствам степенных рядов это необходимо проверить только для положительного целого n, так как можно положить $q^n = y$; затем можно проверить тождество для бесконечного числа значений y. Если теперь обратить верхний индекс в q-номиальной теореме, получится

$$\prod_{k\geq 0}\frac{(1-q^{k+r+1}x)}{(1-q^k x)} = \sum_k \binom{-r-1}{k}_q q^{k(k-1)/2}(-q^{r+1}x)^k = \sum_k \binom{k+r}{k}_q x^k.$$

Эта формула была получена Коши [*Comptes Rendus Acad. Sci. Paris* **17** (1843), 523] и Гейне (Heine) [*Crelle* **34** (1847), 285–328]. Дополнительную информацию можно найти в работе G. Gasper, M. Rahman, *Basic Hypergeometric Series* (Cambridge Univ. Press, 1990).

59. $(n+1)\binom{n}{k} - \binom{n}{k+1}$.

60. $\binom{n+k-1}{k}$. Данную формулу легко запомнить, так как это

$$\frac{n(n+1)\ldots(n+k-1)}{k(k-1)\ldots 1},$$

что аналогично (2), но в числителе идет возрастание, а не убывание значений. Существует остроумный способ доказательства рассматриваемой формулы: достаточно заметить, что нужно подсчитать число целых решений $(a_1,\ldots,a_k)$ неравенств $1\leq a_1\leq a_2\leq\cdots\leq a_k\leq n$. А это равносильно соотношениям $0<a_1<a_2+1<\cdots<a_k+k-1<n+k$. Число решений неравенств

$$0<b_1<b_2<\cdots<b_k<n+k$$

равно числу выборов k различных элементов из множества $\{1, 2,\ldots, n+k-1\}$. (Идея этого доказательства принадлежит Г. Ф. Шерку (H. F. Scherk), *Crelle* **3** (1828), 97. Но самое интересное, что в этом же журнале, **13** (1835), 237, описанный метод был вновь предложен В. А. Ферстеманном (W. A. Förstemann), который заявил: "Можно не сомневаться, что это доказательство известно уже давно, но, как ни странно, я нигде его не нашел, хотя и просмотрел множество работ".)

61. Если a_{mn} — нужная величина, то из соотношений (46) и (47) имеем $a_{mn} = na_{m(n-1)} + \delta_{mn}$. Поэтому в результате получаем $[n\geq m]\,n!/m!$. Эту же формулу легко получить путем обращения соотношения (56).

62. Используйте тождество из упр. 31, выполнив замену $(m,n,r,s,k)\leftarrow(m+k,\ l-k,\ m+n,\ n+l,\ j)$:

$$\sum_k (-1)^k\binom{l+m}{l+k}\binom{m+n}{m+k}\binom{n+l}{n+k}$$

$$= \sum_{j,k}(-1)^k\binom{l+m}{l+k}\binom{l+k}{j}\binom{m-k}{l-k-j}\binom{m+n+j}{m+l}$$

$$= \sum_{j,k}(-1)^k\binom{2l-2j}{l-j+k}\frac{(m+n+j)!}{(2l-2j)!\,j!\,(m-l+j)!\,(n+j-l)!}$$

(мы поменяли знаки факториалов). Теперь сумма по k обратится в нуль (исключение составляет случай $r = l$).

Частный случай данного тождества при $l = m = n$ был опубликован А. К. Диксоном (A. C. Dixon) [*Messenger of Math.* **20** (1891), 79–80], который через 12 лет обобщил эту формулу [*Proc. London Math. Soc.* **35** (1903), 285–289]. Но тем временем Л. Дж. Роджерс (L. J. Rogers) уже опубликовал гораздо более общую формулу [*Proc. London Math. Soc.* **26** (1895), 15–32, §8]. См. также статьи П. А. Мак-Магона (P. A. MacMahon), *Quarterly Journal of Pure and Applied Math.* **33** (1902), 274–288, и Джона Дугалла (John Dougall), *Proc. Edinburgh Math. Society* **25** (1907), 114–132. Соответствующие q-номиальные тождества выглядят следующим образом:

$$\sum_k \binom{m-r+s}{k}_q \binom{n+r-s}{n-k}_q \binom{r+k}{m+n}_q q^{(m-r+s-k)(n-k)} = \binom{r}{m}_q \binom{s}{n}_q,$$

$$\sum_k (-1)^k \binom{l+m}{l+k}_q \binom{m+n}{m+k}_q \binom{n+l}{n+k}_q q^{(3k^2-k)/2} = \frac{(l+m+n)!_q}{l!_q\, m!_q\, n!_q},$$

где $n!_q = \prod_{k=1}^{n}(1 + q + \cdots + q^{k-1})$.

63. См. *CMath*, упр. 5.83 и 5.106.

64. Пусть $f(n, m)$ — это число разбиений множества $\{1, 2, \ldots, n\}$ на m частей. Ясно, что $f(1, m) = \delta_{1m}$. Если $n > 1$, то существует два вида разбиения: (a) элемент n образует одно из множеств разбиения; существует $f(n-1,\, m-1)$ способов построить разбиения такого типа; (b) элемент n появляется в разбиении вместе с другим элементом; существует m способов вставить элемент n в любое m-разбиение множества $\{1, 2, \ldots, n-1\}$. Следовательно, существует $mf(n-1,\, m)$ способов построения подобных разбиений. Отсюда можно заключить, что $f(n, m) = f(n-1,\, m-1) + mf(n-1,\, m)$ и по индукции $f(n, m) = \{{n \atop m}\}$.

65. См. *AMM* **99** (1992), 410–422.

66. Сначала обратите внимание, что $\binom{z}{n+1} \le \binom{y}{n+1}$. Это очевидно, если $z \ge n$, так как $z \le y$; в противном случае $n - 1 \le z \le n$ и, следовательно, $\binom{z}{n+1} \le 0 \le \binom{y}{n+1}$. Поэтому $\binom{z+1}{n+1} = \binom{z}{n+1} + \binom{z}{n} \le \binom{y}{n+1} + \binom{z}{n} = \binom{x}{n+1}$ и мы имеем $x \ge z + 1$.

Теперь можно доказать, что каждый член суммы

$$\binom{x}{n+1} - \binom{y}{n+1} = \sum_{k\ge 0} \binom{z-k}{n-k} t_k, \qquad t_k = \binom{x-z-1+k}{k+1} - \binom{y-z-1+k}{k+1}$$

является неотрицательным Коэффициент $\binom{z-k}{n-k}$ неотрицателен, так как $z \ge n-1$; неотрицателен и $\binom{x-z-1+k}{k+1}$, так как $x \ge z + 1$. Поэтому из $z \le y \le x$ следует, что $\binom{y-z-1+k}{k+1} \le \binom{x-z-1+k}{k+1}$.

Нужный результат очевиден при $x = y$ и $z = n - 1$. В противном случае

$$\binom{x}{n} - \binom{y}{n} - \binom{z}{n-1} = \sum_{k\ge 0} \binom{z-k}{n-1-k} (t_k - \delta_{k0}) = \sum_{k\ge 0} \frac{n-k}{z-n+1} \binom{z-k}{n-k} (t_k - \delta_{k0}),$$

что меньше или равно

$$\sum_{k\ge 0} \frac{n-1}{z-n+1} \binom{z-k}{n-k} (t_k - \delta_{k0}) = \frac{n-1}{z-n+1} \left(\binom{x}{n+1} - \binom{y}{n+1} - \binom{z}{n} \right) = 0,$$

так как $t_0 - 1 = x - y - 1 \le 0$. [L. Lovász, *Combinatorial Problems and Exercises*, Problem 13.31(a).]

67. При $k > 0$ упр 1 2 5–24 дает несколько более точные (но не так легко запоминаемые) оценки сверху $\binom{n}{k} = n^{\underline{k}}/k! \le n^k/k! \le \frac{1}{e}(\frac{ne}{k})^k \le (\frac{ne}{k+1})^k$ Соответствующая оценка снизу выглядит так $\binom{n}{k} \ge (\frac{(n-k)e}{k^2})^k \frac{1}{ek}$

68. Положим $t_k = k\binom{n}{k}p^k(1-p)^{n+1-k}$ Тогда $t_k - t_{k+1} = \binom{n}{k}p^k(1-p)^{n-k}(k-np)$ Поэтому наша сумма равна

$$\sum_{k<\lceil np\rceil}(t_{k+1}-t_k) + \sum_{k\ge\lceil np\rceil}(t_k - t_{k+1}) = 2t_{\lceil np\rceil}$$

[Де Муавр сформулировал это тождество в журнале *Miscellanea Analytica* (1730), 101, для случая, когда np — целое, А Пуанкаре (H Poincaré) опубликовал доказательство общего случая в *Calcul des Probabilités* (1896), 56–60 Интересную историю этого тождества, а также аналогичные формулы можно найти в работе P Diaconis, S Zabell, *Statistical Science* **6** (1991)]

РАЗДЕЛ 1.2.7

1. 0, 1 и 3/2

2. Заменяем каждый член $1/(2^m + k)$ оценкой сверху $1/2^m$

3. $H^{(r)}_{2^m-1} \le \sum_{0\le k<m} 2^k/2^{kr}$, верхняя грань $2^{r-1}/(2^{r-1}-1)$

4. (b) и (c)

5. 9 78760 60360 44382

6. Индукцией с помощью формулы 1 2 6–(46)

7. $T(m+1,n) - T(m,n) = 1/(m+1) - 1/(mn+1) - \quad - 1/(mn+n) \le 1/(m+1) - (1/(mn+n) + \quad + 1/(mn+n)) = 1/(m+1) - n/(mn+n) = 0$ Максимум достигается при $m = n = 1$, а минимум — при очень больших m и n Согласно (3) точная нижняя грань равна γ, которая в действительности никогда не достигается Обобщение этого результата можно найти в *AMM* **70** (1963), 575–577

8. По формуле Стирлинга $\ln n!$ приблизительно равен $(n+\frac{1}{2})\ln n - n + \ln\sqrt{2\pi}$, сумма $\sum_{k=1}^n H_k$ приблизительно равна $(n+1)\ln n - n(1-\gamma) + (\gamma + \frac{1}{2})$, разность приблизительно равна $\gamma n + \frac{1}{2}\ln n +$ 158

9. $-1/n$

10. Разбиваем левую часть на две суммы и во второй сумме заменяем k на $k+1$

11. $2 - H_n/n - 1/n$ *для* $n > 0$

12. Значение 1 000 верно с точностью до более чем трехсотого десятичного знака

13. Как и при доказательстве теоремы A, воспользуемся методом индукции Есть и другой способ продифференцировать по x и вычислить при $x = 1$

14. См раздел 1 2 3, пример 2 Вторая сумма равна $\frac{1}{2}(H^2_{n+1} - H^{(2)}_{n+1})$

15. $\sum_{j=1}^n (1/j)\sum_{k=j}^n H_k$ можно просуммировать по формулам, приведенным в тексте В результате получим $(n+1)H_n^2 - (2n+1)H_n + 2n$

16. $H_{2n-1} - \frac{1}{2}H_{n-1}$

17. *Первое решение* (элементарное) (Положим знаменатель равным $(p-1)!$, что кратно истинному знаменателю, но не кратно p Тогда достаточно показать, что соответствующий числитель, $(p-1)!/1 + (p-1)!/2 + \quad + (p-1)!/(p-1)$, *является* кратным p $(p-1)!/k \equiv (p-1)!\,k'$ по модулю p, где k' определяется из соотношения $kk' \bmod p = 1$ Множество $\{1', 2', \quad, (p-1)'\}$ — это просто множество $\{1, 2, \quad, p-1\}$, поэтому числитель сравним с $(p-1)!\,(1 + 2 + \quad + p - 1) \equiv 0$

Второе решение (более сложное). Из упр. 4.6.2–6 имеем $x^{\bar{p}} \equiv x^p - x$ (по модулю p); тогда из упр. 1.2.6–32 получаем $\left[{p \atop k}\right] \equiv \delta_{kp} - \delta_{k1}$. А теперь применим упр. 6.

Известно, что числитель H_{p-1} на самом деле кратен p^2 при $p > 3$; см. работу Hardy, Wright, *An Introduction to the Theory of Numbers*, Section 7.8.

18. Если $n = 2^k m$, где m нечетно, то сумма равна $2^{2k} m_1/m_2$, где и m_1, и m_2 нечетны. [*AMM* **67** (1960), 924–925.]

19. Только при $n = 0$, $n = 1$. Для $n \geq 2$ положим $k = \lfloor \lg n \rfloor$. Существует ровно один член, знаменатель которого равен 2^k, поэтому $2^{k-1} H_n - \frac{1}{2}$ является суммой членов, в знаменателе которых содержатся только простые числа. Если бы H_n было целым, то $2^{k-1} H_n - \frac{1}{2}$ имело бы знаменатель, равный 2.

20. Разлагаем подынтегральное выражение и интегрируем почленно. См. также *AMM* **69** (1962), 239, и статью H. W. Gould, *Mathematics Magazine* **34** (1961), 317–321.

21. $H_{n+1}^2 - H_{n+1}^{(2)}$.

22. $(n+1)(H_n^2 - H_n^{(2)}) - 2n(H_n - 1)$.

23. $\Gamma'(n+1)/\Gamma(n+1) = 1/n + \Gamma'(n)/\Gamma(n)$, так как $\Gamma(x+1) = x\Gamma(x)$. Отсюда $H_n = \gamma + \Gamma'(n+1)/\Gamma(n+1)$. Функция $\psi(x) = \Gamma'(x)/\Gamma(x) = H_{x-1} - \gamma$ называется *пси-функцией* или *дигамма-функцией*. Некоторые значения для рациональных x приведены в приложении А.

24. Получаем

$$x \lim_{n\to\infty} e^{(H_n - \ln n)x} \prod_{k=1}^{n} \left(\left(1 + \frac{x}{k}\right) e^{-x/k}\right) = \lim_{n\to\infty} \frac{x(x+1)\ldots(x+n)}{n^x n!}.$$

Замечание. Следовательно, обобщенная величина H_n, рассмотренная в предыдущем упражнении, равна $H_x^{(r)} = \sum_{k\geq 0}(1/(k+1)^r - 1/(k+1+x)^r)$ при $r = 1$. Эту же идею можно использовать для более высоких значений r. Наше бесконечное произведение сходится для всех комплексных x.

РАЗДЕЛ 1.2.8

1. Через k месяцев будет F_{k+2} пар, значит, через год — $F_{14} = 377$ пар.

2. $\ln(\phi^{1000}/\sqrt{5}) = 1000 \ln \phi - \frac{1}{2} \ln 5 = 480.40711$. $\log_{10} F_{1000}$ равен предыдущей величине, умноженной на $1/(\ln 10)$, т. е. 208.64. Следовательно, F_{1000} — это число, состоящее из 209 цифр, первой из которых является 4.

4. 0, 1, 5; после этого функция F_n возрастает слишком быстро.

5. 0, 1, 12.

6. По индукции. (Это равенство выполняется также для *отрицательных* n; см. упр. 8.)

7. Если d является собственным делителем n, то F_d делит F_n. Далее, F_d больше единицы и меньше F_n при условии, что d больше 2. Единственное не простое число, которое не имеет собственного делителя, большего 2, — это $n = 4$. Следовательно, единственным исключением является $F_4 = 3$.

8. $F_{-1} = 1$; $F_{-2} = -1$. Индукцией по n можно доказать, что $F_{-n} = (-1)^{n+1} F_n$.

9. Не выполняется (15). Остальные соотношения справедливы; это можно установить индукцией "назад", т. е. показать, что предположение верно для $n - 1$, если оно верно для n и бóльших значений.

10. Если n четно, то оно больше, а если n нечетно, то меньше (см. формулу (14)).

11. По индукции (см. упр. 9). Это частный случай упр. 13, (а).

12. Если $\mathcal{G}(z) = \sum \mathcal{F}_n z^n$, $(1 - z - z^2)\mathcal{G}(z) = z + F_0 z^2 + F_1 z^3 + \cdots = z + z^2 G(z)$. Отсюда $\mathcal{G}(z) = G(z) + zG(z)^2$; из (17) находим $\mathcal{F}_n = ((3n+3)/5)F_n - (n/5)F_{n+1}$.

13. (a) $a_n = rF_{n-1} + sF_n$. (b) Поскольку $(b_{n+2}+c) = (b_{n+1}+c)+(b_n+c)$, можем рассмотреть новую последовательность $b'_n = b_n + c$. Применяя п. (a) к b'_n, получим $cF_{n-1} + (c+1)F_n - c$.

14. $a_n = F_{m+1}F_{n-1} + (F_{m+2}+1)F_n - \binom{n}{m} - \binom{n+1}{m-1} - \cdots - \binom{n+m}{0}$.

15. $c_n = xa_n + yb_n + (1 - x - y)F_n$.

16. F_{n+1}. Доказываем по индукции и применяем соотношение

$$\binom{n+1-k}{k} = \binom{n-k}{k} + \binom{(n-1)-(k-1)}{k-1}.$$

17. Воспользуемся тем, что $(x^{n+k} - y^{n+k})(x^{m-k} - y^{m-k}) - (x^n - y^n)(x^m - y^m)$ равно $(xy)^n(x^{m-n-k} - y^{m-n-k})(x^k - y^k)$. В этом соотношении положим $x = \phi$, $y = \hat{\phi}$ и разделим его на $(\sqrt{5})^2$.

18. Да, так как это F_{2n+1}.

19. Пусть $u = \cos 72°$, $v = \cos 36°$. Имеем $u = 2v^2 - 1$, $v = 1 - 2\sin^2 18° = 1 - 2u^2$. Отсюда $u + v = 2(v^2 - u^2)$, т. е. $1 = 2(v - u) = 2v - 4v^2 + 2$. Следовательно, $v = \frac{1}{2}\phi$. (Кроме того, $u = \frac{1}{2}\phi^{-1}$, $\sin 36° = \frac{1}{2}5^{1/4}\phi^{-1/2}$, $\sin 72° = \frac{1}{2}5^{1/4}\phi^{1/2}$.)

20. $F_{n+2} - 1$.

21. Умножаем на $x^2 + x - 1$ и находим ответ: $(x^{n+1}F_{n+1} + x^{n+2}F_n - x)/(x^2 + x - 1)$. Если знаменатель обращается в нуль, т. е. x равно $1/\phi$ или $1/\hat{\phi}$, то решением будет

$$((n+1)x^n F_{n+1} + (n+2)x^{n+1}F_n - 1)/(2x+1).$$

22. F_{m+2n}; в следующем упражнении положите $t = 2$.

23. $\frac{1}{\sqrt{5}}\sum_k \binom{n}{k}(\phi^k F_t^k F_{t-1}^{n-k}\phi^m - \hat{\phi}^k F_t^k F_{t-1}^{n-k}\hat{\phi}^m)$

$$= \frac{1}{\sqrt{5}}(\phi^m(\phi F_t + F_{t-1})^n - \hat{\phi}^m(\hat{\phi}F_t + F_{t-1})^n) = F_{m+tn}.$$

24. F_{n+1} (разложите определитель по элементам первой строки).

25. $2^n\sqrt{5}\,F_n = (1+\sqrt{5})^n - (1-\sqrt{5})^n$.

26. По теореме Ферма $2^{p-1} \equiv 1$; теперь используем предыдущее упражнение и упр. 1.2.6–10(b).

27. Для $p = 2$ утверждение верно. Для остальных p справедливо соотношение $F_{p-1}F_{p+1} - F_p^2 = -1$. Тогда из предыдущего упражнения и по теореме Ферма получаем: $F_{p-1}F_{p+1} \equiv 0$ (по модулю p). Только один из этих сомножителей может быть кратным p, поскольку $F_{p+1} = F_p + F_{p-1}$.

28. $\hat{\phi}^n$. *Замечание.* Решением рекуррентных соотношений $a_{n+1} = Aa_n + B^n$, $a_0 = 0$ является

$$a_n = (A^n - B^n)/(A - B), \text{ если } A \neq B, \qquad a_n = nA^{n-1}, \text{ если } A = B.$$

29. (a)

$\binom{n}{0}_{\mathcal{F}}$	$\binom{n}{1}_{\mathcal{F}}$	$\binom{n}{2}_{\mathcal{F}}$	$\binom{n}{3}_{\mathcal{F}}$	$\binom{n}{4}_{\mathcal{F}}$	$\binom{n}{5}_{\mathcal{F}}$	$\binom{n}{6}_{\mathcal{F}}$
1	0	0	0	0	0	0
1	1	0	0	0	0	0
1	1	1	0	0	0	0
1	2	2	1	0	0	0
1	3	6	3	1	0	0
1	5	15	15	5	1	0
1	8	40	60	40	8	1

(b) следует из (6). [É. Lucas, *Amer. J. Math.* **1** (1878), 201–204.]

30. Доказательство проводим индукцией по m; при $m = 1$ утверждение очевидно.

(a) $\sum_k \binom{m}{k}_{\mathcal{F}} (-1)^{\lceil (m-k)/2 \rceil} F_{n+k}^{m-2} F_k = F_m \sum_k \binom{m-1}{k-1}_{\mathcal{F}} (-1)^{\lceil (m-k)/2 \rceil} F_{n+k}^{m-2} = 0.$

(b) $\sum_k \binom{m}{k}_{\mathcal{F}} (-1)^{\lceil (m-k)/2 \rceil} F_{n+k}^{m-2} (-1)^k F_{m-k}$

$$= (-1)^m F_m \sum_k \binom{m-1}{k}_{\mathcal{F}} (-1)^{\lceil (m-1-k)/2 \rceil} F_{n+k}^{m-2} = 0.$$

(c) Так как $(-1)^k F_{m-k} = F_{k-1} F_m - F_k F_{m-1}$ и $F_m \neq 0$, из (a) и (b) следует, что $\sum_k \binom{m}{k}_{\mathcal{F}} (-1)^{\lceil (m-k)/2 \rceil} F_{n+k}^{m-2} F_{k-1} = 0$.

(d) Поскольку $F_{n+k} = F_{k-1} F_n + F_k F_{n+1}$, результат следует из (a) и (c). Этот результат можно также доказать в более общем виде, если воспользоваться q-номиальной теоремой из упр. 1.2.6–58. [См. Dov Jarden, *Recurring Sequences*, 2nd ed. (Jerusalem, 1966), 30–33; J. Riordan, *Duke Math. J.* **29** (1962), 5–12.]

31. Используйте упр. 8 и 11.

32. По модулю F_n последовательность Фибоначчи выглядит так: $0, 1, \ldots, F_{n-1}, 0, F_{n-1}, -F_{n-2}, \ldots$.

33. Заметим, что $\cos z = \frac{1}{2}(e^{iz} + e^{-iz}) = -i/2$ для этого конкретного z, и воспользуемся тем, что $\sin(n+1)z + \sin(n-1)z = 2 \sin nz \cos z$ для всех z.

34. Сначала докажем, что единственно возможным значением F_{k_1} является наибольшее число Фибоначчи, которое меньше или равно n. Отсюда $n - F_{k_1}$ меньше, чем F_{k_1-1}, и по индукции получаем, что существует единственное представление $n - F_{k_1}$. Это доказательство во многом аналогично доказательству теоремы о единственности разложения целого числа на простые множители. Система чисел Фибоначчи была предложена Э. Зекендорфом (E. Zeckendorf) [см. *Simon Stevin* **29** (1952), 190–195; *Bull. Soc. Royale des Sciences de Liège* **41** (1972), 179–182]; более общие результаты приведены в упр. 5.4.2–10.

35. См. G. M. Bergman, *Mathematics Magazine* **31** (1957), 98–110. Чтобы представить $x > 0$, найдите наибольшее k, для которого $\phi^k \leq x$, и представьте x как ϕ^k плюс представление $x - \phi^k$.

Представление для неотрицательных целых чисел можно получить также с помощью следующих рекуррентных соотношений, которые справедливы для всех целых чисел, начиная с 0 и 1 (представление которых тривиально). Пусть $L_n = \phi^n + \hat{\phi}^n = F_{n+1} + F_{n-1}$. Тогда представлением $L_{2n} + m$ для $0 \leq m \leq L_{2n-1}$ и $n \geq 1$ будет $\phi^{2n} + \phi^{-2n}$ плюс представление m. Представлением $L_{2n+1} + m$ для $0 < m < L_{2n}$ и $n \geq 0$ будет $\phi^{2n+1} + \phi^{-2n-2}$ плюс представление $m - \phi^{-2n}$. Последний результат получен с помощью соотношения $\phi^k - \phi^{k-2j} = \phi^{k-1} + \phi^{k-3} + \cdots + \phi^{k-2j+1}$. Оказывается, что все строки α, состоящие из нулей и единиц (такие, что α начинается с 1 и не имеет соседней единицы), встречаются слева от разделяющей точки в представлении ровно одного положительного целого числа. Исключение составляют строки, которые заканчиваются $10^{2k}1$, но они никогда не встречаются в подобных представлениях.

36. Рассмотрим бесконечную строку S_∞, первые F_n букв которой для любого $n > 1$ образуют строку S_n. В этой строке нет ни удвоения буквы a, ни утроения буквы b. В строке S_n содержится F_{n-2} букв a и F_{n-1} букв b. Если выразить $m - 1$ с помощью системы чисел Фибоначчи (как в упр. 34), то a будет m-й буквой строки S_∞ тогда и только тогда, когда $k_r = 2$. С другой стороны, b будет k-й буквой строки S_∞ тогда и только тогда, когда $\lfloor (k+1)\phi^{-1} \rfloor - \lfloor k\phi^{-1} \rfloor = 1$. Поэтому количество букв b среди первых k букв равно $\lfloor (k+1)\phi^{-1} \rfloor$. Кроме того, b будет k-й буквой тогда и только тогда, когда $k = \lfloor m\phi \rfloor$ для некоторого целого положительного m. Эту последовательность изучали Жан Бернулли III

(Jean Bernoulli III) в 18 веке, А. А. Марков — в 19 веке, а впоследствии — многие другие математики; см. K. B. Stolarsky, *Canadian Math. Bull.* **19** (1976), 473–482

37. [*Fibonacci Quart.* **1** (December, 1963), 9–12.] Рассмотрим систему чисел Фибоначчи из упр. 34. Если в ней $n = F_{k_1} + \cdots + F_{k_r} > 0$, то положим $\mu(n) = F_{k_r}$. Пусть $\mu(0) = \infty$. Докажем следующие свойства. (A) Если $n > 0$, то $\mu(n - \mu(n)) > 2\mu(n)$. *Доказательство.* $\mu(n - \mu(n)) = F_{k_r-1} \geq F_{k_r+2} > 2F_{k_r}$, так как $k_r \geq 2$. (B) Если $0 < m < F_k$, то $\mu(m) \leq 2(F_k - m)$. *Доказательство.* Пусть $\mu(m) = F_j$; $m \leq F_{k-1} + F_{k-3} + \cdots + F_{j+(k-1-j) \bmod 2} = -F_{j-1+(k-1-j) \bmod 2} + F_k \leq -\frac{1}{2}F_j + F_k$. (C) Если $0 < m < \mu(n)$, то $\mu(n - \mu(n) + m) \leq 2(\mu(n) - m)$. *Доказательство.* Следует из (B). (D) Если $0 < m < \mu(n)$, то $\mu(n - m) \leq 2m$. *Доказательство.* В (C) положим $m = \mu(n) - m$.

Теперь докажем, что если имеется n фишек и на следующем ходе можно взять максимум q фишек, то данный ход будет выигрышным тогда и только тогда, когда $\mu(n) \leq q$. *Доказательство.* (a) Если $\mu(n) > q$, то после каждого хода получается положение n', q', где $\mu(n') \leq q'$. [Это следует из (D); см выше.] (b) Если $\mu(n) \leq q$, мы можем либо выиграть на этом ходу (если $q \geq n$), либо сделать ход, который приведет к положению n', q', где $\mu(n') > q'$. [Это следует из (A); см. выше. Делая ход, мы должны взять $\mu(n)$ фишек.] Можно легко показать, что если $n = F_{k_1} + \cdots + F_{k_r}$, то выигрышные ходы состоят в том, чтобы брать $F_{k_j} + \cdots + F_{k_r}$ фишек для некоторого j, $1 \leq j \leq r$, при условии, что $j = 1$ либо $F_{k_{j-1}} > 2(F_{k_j} + \cdots + F_{k_r})$.

Для числа 1 000 имеем следующее представление Фибоначчи: $987 + 13$. Существует *единственный* удачный ход, позволяющий добиться победы, — взять 13 фишек. Заметим, что первый игрок всегда имеет шансы выиграть; исключение составляет случай, когда n не является числом Фибоначчи.

Решение для более общих игр подобного типа было получено А. Швенком (A. Schwenk) [*Fibonacci Quarterly* **8** (1970), 225–234].

39. $(3^n - (-2)^n)/5$.

40. Индукцией по m докажем, что $f(n) = m$ для $F_m < n \leq F_{m+1}$. Во-первых, $f(n) \leq \max(1 + f(F_m), 2 + f(n - F_m)) = m$. Во-вторых, если $f(n) < m$, существует некоторое $k < n$, такое, что $1 + f(k) < m$ (отсюда $k \leq F_{m-1}$) и $2 + f(n-k) < m$ (отсюда $n - k \leq F_{m-2}$). Следовательно, $n \leq F_{m-1} + F_{m-2}$. [Таким образом, деревья Фибоначчи, которые будут определены в разделе 6.2.1, минимизируют максимальную стоимость внутреннего пути от корня до листа, если правая ветвь стоит вдвое дороже, чем левая.]

41. $F_{k_1+1} + \cdots + F_{k_r+1} = \phi n + (\hat{\phi}^{k_1} + \cdots + \hat{\phi}^{k_r})$ — это целое число, а величина в скобках лежит между $\hat{\phi}^3 + \hat{\phi}^5 + \cdots = \phi^{-1} - 1$ и $\hat{\phi}^2 + \hat{\phi}^4 + \cdots = \phi^{-1}$. Аналогично $F_{k_1-1} + \cdots + F_{k_r-1} = \phi^{-1}n + (\hat{\phi}^{k_1} + \cdots + \hat{\phi}^{k_r}) = f(\phi^{-1}n)$. [Такое смещение чисел Фибоначчи представляет собой удобный способ перевода в уме миль в километры и наоборот; см *CMath*, §6.6.]

42. [*Fibonacci Quarterly* **6** (1968), 235–244.] Если существует такое представление, то имеем

$$mF_{N-1} + nF_N = F_{k_1+N} + F_{k_2+N} + \cdots + F_{k_r+N} \qquad (*)$$

для всех целых N. Следовательно, существование двух различных представлений противоречило бы упр. 34.

Обратно, существование таких совместных представлений для всех неотрицательных m и n можно доказать по индукции. Но гораздо интереснее воспользоваться предыдущим упражнением и доказать, что такие совместные представления существуют, возможно, и для отрицательных целых m и n тогда и только тогда, когда $m + \phi n \geq 0$. Пусть N достаточно велико для того, чтобы выполнялось неравенство $|m\hat{\phi}^{N-1} + n\hat{\phi}^N| < \phi^{-2}$, и представим $mF_{N-1} + nF_N$ с помощью соотношения $(*)$. Тогда $mF_N + nF_{N+1} = \phi(mF_{N-1} + nF_N) + (m\hat{\phi}^{N-1} + n\hat{\phi}^N) = f(\phi(mF_{N-1} + nF_N)) = F_{k_1+N+1} + \cdots + F_{k_r+N+1}$. Отсюда следует, что $(*)$ выполняется для всех N. Теперь положим $N = 0$ и $N = 1$.

РАЗДЕЛ 1.2.9

1. $1/(1-2z)+1/(1-3z)$

2. Она следует из (6), так как $\binom{n}{k}=n^{\underline{k}}/k!$

3. $G'(z)=\ln(1/(1-z))/(1-z)^2+1/(1-z)^2$ Отсюда и из формулы для $G(z)/(1-z)$ следует, что $\sum_{k=1}^{n-1}H_k=nH_n-n$, это согласуется с 1 2 7–(8)

4. Положим $t=0$

5. Согласно (11) и (22) коэффициент при z^k равен

$$(n-1)!\sum_{0\le j<k}\left\{{j\atop n-1}\right\}\binom{k}{j}$$

Теперь применим 1 2 6–(46) и 1 2 6–(52) (или продифференцируем и используем 1 2 6–(46))

6. $(\ln(1/(1-z)))^2$ Производная равна удвоенной производящей функции для гармонических чисел, поэтому сумма равна $2H_{n-1}/n$

8. $1/((1-z)(1-z^2)(1-z^3)\quad)$ [Исторически это один из первых случаев применения производящих функций Интересный рассказ об исследовании этой производящей функции, которое проводил Л Эйлер в 18 веке, можно найти в книге G Pólya, *Induction and Analogy in Mathematics* (Princeton Princeton University Press, 1954), Chapter 6 (Пойа Д *Математика и правдоподобные рассуждения*, т 1 *Индукция и аналогия в математике* (М Изд-во иностр лит , 1957)]

9. $\frac{1}{24}S_1^4+\frac{1}{4}S_1^2S_2+\frac{1}{8}S_2^2+\frac{1}{3}S_1S_3+\frac{1}{4}S_4$

10. $G(z)=(1+x_1z)\quad(1+x_nz)$ Логарифмируя, как при выводе соотношения (38), получим те же формулы, но соотношение (24) заменит (17) Ответ будет точно таким же, только $S_2,S_4,S_6,\quad$ заменятся $-S_2,\ -S_4,\ -S_6,\quad$ Имеем $a_1=S_1$, $a_2=\frac{1}{2}S_1^2-\frac{1}{2}S_2$, $a_3=\frac{1}{6}S_1^3-\frac{1}{2}S_1S_2+\frac{1}{3}S_3$, $a_4=\frac{1}{24}S_1^4-\frac{1}{4}S_1^2S_2+\frac{1}{8}S_2^2+\frac{1}{3}S_1S_3-\frac{1}{4}S_4$ (см упр 9) Аналогичное (39) рекуррентное соотношение выглядит следующим образом $na_n=S_1a_{n-1}-S_2a_{n-2}+$

Замечание Формулы, которые дает это рекуррентное соотношение, называются *тождествами Ньютона*, так как впервые они были опубликованы Исааком Ньютоном в работе *Arithmetica Universalis* (1707), см D J Struik *Source Book in Mathematics* (Harvard University Press, 1969), 94–95

11. Так как $\sum_{m\ge1}S_mz^m/m=\ln G(z)=\sum_{k\ge1}(-1)^{k-1}(h_1z+h_2z^2+\quad)^k/k$, нужный коэффициент равен $(-1)^{k_1+k_2+\quad+k_m-1}m(k_1+k_2+\quad+k_m-1)!/k_1!k_2!\quad k_m!$ [Умножаем на $(-1)^{m-1}$, чтобы получить коэффициент при $a_1^{k_1}a_2^{k_2}\quad a_m^{k_m}$ в формуле, где S_m выражено через a_m из упр 10 Альбер Жирар (Albert Girard) сформулировал соотношения для S_1, S_2, S_3 и S_4, выразив их через a_1, a_2, a_3 и a_4 Эти формулы приводятся почти в самом конце его работы *Invention Nouvelle en Algébre* (Amsterdam, 1629), так родилась теория симметричных функций]

12. $\displaystyle\sum_{m\ n\ge0}a_{mn}w^mz^n=\sum_{m\ n\ge0}\binom{n}{m}w^mz^n=\sum_{n\ge0}(1+w)^nz^n=1/(1-z-wz)$

13. $\int_n^{n+1}e^{-st}f(t)\,dt=(a_0+\quad+a_n)(e^{-sn}-e^{-s(n+1)})/s$ Сложив эти выражения для всех n, получим $\mathbf{L}f(s)=G(e^{-s})/s$

14. См упр 1 2 6-38

15. $G_n(z)=G_{n-1}(z)+zG_{n-2}(z)+\delta_{n0}$, поэтому $H(w)=1/(1-w-zw^2)$ Отсюда окончательно находим

$$G_n(z)=\left(\left(\frac{1+\sqrt{1+4z}}{2}\right)^{n+1}-\left(\frac{1-\sqrt{1+4z}}{2}\right)^{n+1}\right)\Big/\sqrt{1+4z},\quad\text{если } z\ne-\tfrac{1}{4},$$

$G_n(-\frac{1}{4})=(n+1)/2^n$ при $n\ge0$

16. $G_{nr}(z) = (1 + z + \cdots + z^r)^n = \left(\dfrac{1 - z^{r+1}}{1 - z}\right)^n$. [Обратите внимание на случай $r = \infty$.]

17. $\displaystyle\sum_k \binom{-w}{k}(-z)^k = \sum_k \frac{w(w+1)\dots(w+k-1)}{k(k-1)\dots 1} z^k = \sum_{n,k} \begin{bmatrix} k \\ n \end{bmatrix} z^k w^n / k!.$

(Есть и другой способ: записать эту функцию в виде $e^{w \ln(1/(1-z))}$ и разложить сначала по степеням w.)

18. (a) Для фиксированного n и переменного r согласно (27) производящая функция равна

$$G_n(z) = (1+z)(1+2z)\dots(1+nz) = z^{n+1}\left(\frac{1}{z}\right)\left(\frac{1}{z}+1\right)\left(\frac{1}{z}+2\right)\dots\left(\frac{1}{z}+n\right)$$
$$= \sum_k \begin{bmatrix} n+1 \\ k \end{bmatrix} z^{n+1-k}$$

(см. формулу (27)). Отсюда получаем ответ: $\left[\begin{smallmatrix} n+1 \\ n+1-r \end{smallmatrix}\right]$. (b) Аналогично согласно (28) производящая функция равна

$$\frac{1}{1-z} \cdot \frac{1}{1-2z} \cdot \dots \cdot \frac{1}{1-nz} = \sum_k \begin{Bmatrix} k \\ n \end{Bmatrix} z^{k-n},$$

поэтому получаем ответ: $\left\{\begin{smallmatrix} n+r \\ n \end{smallmatrix}\right\}$.

19. $\sum_{n\ge 1}(1/n - 1/(n+p/q))x^{p+nq} = \sum_{k=0}^{q-1} \omega^{-kp} \ln(1-\omega^k x) - x^p \ln(1-x^q) + \frac{q}{p}x^p = f(x) + g(x)$, где $\omega = e^{2\pi\imath/q}$ и

$$f(x) = \sum_{k=1}^{q-1} \omega^{-kp} \ln(1 - \omega^k x), \quad g(x) = (1-x^p)\ln(1-x) + \frac{q}{p}x^p - x^p \ln\frac{1-x^q}{1-x}.$$

Теперь получим $\lim_{x\to 1-} g(x) = q/p - \ln q$. Из тождества

$$\ln(1 - e^{\imath\theta}) = \ln\left(2e^{\imath(\theta-\pi)/2}\,\frac{e^{\imath\theta/2} - e^{-\imath\theta/2}}{2i}\right) = \ln 2 + \tfrac{1}{2}i(\theta - \pi) + \ln\sin\frac{\theta}{2}$$

можно записать $f(x) = A + B$, где

$$A = \sum_{k=1}^{q-1} \omega^{-kp}\left(\ln 2 - \frac{i\pi}{2} + \frac{ik\pi}{q}\right) = -\ln 2 + \frac{i\pi}{2} + \frac{i\pi}{(\omega^{-p}-1)};$$
$$B = \sum_{k=1}^{q-1} \omega^{-kp} \ln\sin\frac{k}{q}\pi = \sum_{0<k<q/2} (\omega^{-kp} + \omega^{-(q-k)p}) \ln\sin\frac{k}{q}\pi$$
$$= 2\sum_{0<k<q/2} \cos\frac{2pk}{q}\pi \cdot \ln\sin\frac{k}{q}\pi.$$

Окончательно получаем

$$\frac{i}{2} + \frac{i}{(\omega^{-p}-1)} = \frac{i}{2}\left(\frac{1+\omega^p}{1-\omega^p}\right) = \frac{i}{2}\left(\frac{\omega^{p/2}+\omega^{-p/2}}{\omega^{p/2}-\omega^{-p/2}}\right) = \frac{1}{2}\cot\frac{p}{q}\pi.$$

[Гаусс вывел эту формулу в §33 своей монографии, посвященной гипергеометрическим рядам (соотношение [75]), но доказательство было недостаточно строгим; Абель обосновал данный результат в работе *Crelle* 1 (1826), 314–315.]

20. $c_{mk} = k!\left\{\begin{smallmatrix} m \\ k \end{smallmatrix}\right\}$ согласно соотношению 1.2.6-(45).

21. Находим $z^2G'(z)+zG(z)=G(z)-1$. Решением этого дифференциального уравнения будет $G(z)=(-1/z)e^{-1/z}(E_1(-1/z)+C)$, где $E_1(x)=\int_z^\infty e^{-t}\,dt/t$, а C — константа. Эта функция "очень плохо себя ведет" в окрестности точки $z=0$, и $G(z)$ нельзя разложить в степенной ряд. Действительно, так как функция $\sqrt[n]{n!}\approx n/e$ не ограничена, ряд в представлении производящей функции расходится. Однако при $z<0$ для данной функции существует асимптотическое представление. [См. K. Knopp, *Infinite Sequences and Series* (Dover, 1956), Section 66.]

22. $G(z)=(1+z)^r(1+z^2)^r(1+z^4)^r(1+z^8)^r\ldots=(1-z)^{-r}$. Отсюда следует, что исходная сумма равна $\binom{r+n-1}{n}$.

23. При $m=1$ получаем биномиальную теорему, где $f_1(z)=z$, а $g_1(z)=1+z$. При $m\ge 1$ можно увеличить m на 1, если заменить z_m на $z_m(1+z_{m+1}^{-1})$ и положить $f_{m+1}(z_1,\ldots,z_{m+1})=z_{m+1}f_m(z_1,\ldots,z_{m-1},z_m(1+z_{m+1}^{-1}))$, $g_{m+1}(z_1,\ldots,z_{m+1})=z_{m+1}g_m(z_1,\ldots,z_{m-1},z_m(1+z_{m+1}^{-1}))$. Тогда $g_2(z_1,z_2)=z_1+z_2+z_1z_2$ и

$$\frac{g_m(z_1,\ldots,z_m)}{f_m(z_1,\ldots,z_m)}=1+\cfrac{z_1^{-1}}{1+\cfrac{z_2^{-1}}{1+\cfrac{\ddots}{1+z_m^{-1}}}}.$$

Оба многочлена, f_m и g_m, удовлетворяют одному и тому же рекуррентному соотношению $f_m=z_mf_{m-1}+z_{m-1}f_{m-2}$, $g_m=z_mg_{m-1}+z_{m-1}g_{m-2}$ с начальными условиями $f_{-1}=0$, $f_0=g_{-1}=g_0=z_0=1$. Отсюда следует, что g_m — сумма всех членов, которые можно получить, начиная с $z_1\ldots z_m$ и вычеркивая нули или несоседние коэффициенты; существует F_{m+2} способов сделать это. Точно так же можно интерпретировать f_m, только z_1 должно остаться. В п. (b) мы будем иметь дело с многочленом $h_m=z_mg_{m-1}+z_{m-1}f_{m-2}$. Это сумма всех членов, получаемых из $z_1\ldots z_m$ путем вычеркивания коэффициентов, которые не являются соседними *циклически*. Например, $h_3=z_1z_2z_3+z_1z_2+z_1z_3+z_2z_3$.

(b) Согласно п. (a) $S_n(z_1,\ldots,z_{m-1},z)=[z_m^n]\sum_{r=0}^n z^rz_m^{n-r}f_m^{n-r}g_m^r$. Отсюда

$$S_n(z_1,\ldots,z_m)=\sum_{0\le s\le r\le n}\binom{r}{s}\binom{n-r}{s}a^{r-s}b^sc^sd^{n-r-s},$$

где $a=z_mg_{m-1}$, $b=z_{m-1}g_{m-2}$, $c=z_mf_{m-1}$, $d=z_{m-1}f_{m-2}$. Умножая это равенство на z^n и суммируя сначала по n, затем по r и наконец по s, получим выражение в замкнутом виде:

$$S_n(z_1,\ldots,z_m)=[z^n]\frac{1}{(1-az)(1-dz)-bcz^2}=\frac{\rho^{n+1}-\sigma^{n+1}}{\rho-\sigma},$$

где $1-(a+d)z+(ad-bc)z^2=(1-\rho z)(1-\sigma z)$. Здесь $a+d=h_m$, а $ad-bc$ можно упростить до вида $(-1)^mz_1\ldots z_m$. [Дополнительно мы получили рекуррентное соотношение $S_n=h_mS_{n-1}-(-1)^mz_1\ldots z_mS_{n-2}$, которое не так просто вывести без помощи производящих функций.]

(c) Пусть $\rho_1=(z+\sqrt{z^2+4z})/2$ и $\sigma_1=(z-\sqrt{z^2+4z})/2$ — корни при $m=1$; тогда $\rho_m=\rho_1^m$ и $\sigma_m=\sigma_1^m$.

Карлитц использовал этот результат для вывода следующего удивительного факта. Характеристический многочлен $\det(xI-A)$ матрицы размера $n\times n$

$$A=\begin{pmatrix}0&0&\ldots&0&\binom{0}{0}\\0&0&\ldots&\binom{1}{0}&\binom{1}{1}\\\vdots&\vdots&\ddots&\vdots&\vdots\\\binom{n-1}{0}&\binom{n-1}{1}&\ldots&\binom{n-1}{n-2}&\binom{n-1}{n-1}\end{pmatrix},$$

состоящей из "расположенных по правой стороне биномиальных коэффициентов", равен $\sum_k \binom{n}{k}_{\mathcal{F}} (-1)^{\lceil (n-k)/2 \rceil} x^k$, где $\binom{n}{k}_{\mathcal{F}}$ — фибономиальные коэффициенты (см. упр. 1.2.8–30). Используя аналогичные методы, он также показал, что

$$\sum_{k_1,\ldots,k_m \ge 0} \binom{k_1+k_2}{k_1} \binom{k_2+k_3}{k_2} \cdots \binom{k_m+k_1}{k_m} z_1^{k_1} \ldots z_m^{k_m} = \frac{1}{\sqrt{z_1^2 \ldots z_m^2\, h_m(-z_1^{-1}, \ldots, -z_m^{-1})^2 - 4z_1 \ldots z_m}}.$$

[*Collectanea Math.* **27** (1965), 281–296.]

24. Обе части тождества равны $\sum_k \binom{m}{k} [z^n](zG(z))^k$. Для $G(z) = 1/(1-z)$ тождество принимает вид $\sum_k \binom{m}{k}\binom{n-1}{n-k} = \binom{m+n-1}{n}$ (см. соотношение 1.2.6–(21)), а для $G(z) = (e^z - 1)/z$ — $\sum_k m^{\underline{k}} \left\{ {n \atop k} \right\} = m^n$ (см. соотношение 1.2.6–(45)).

25. $\sum_k [w^k](1-2w)^n [z^n] z^k (1+z)^{2n-2k} = [z^n](1+z)^{2n} \sum_k [w^k](1-2w)^n (z/(1+z)^2)^k$, а это равно $[z^n](1+z)^{2n}(1-2z/(1+z)^2)^n = [z^n](1+z^2)^n = \binom{n}{n/2}$[$n$ четное]. Аналогично находим $\sum_k \binom{n}{k}\binom{2n-2k}{n-k}(-4)^k = (-1)^n \binom{2n}{n}$. Множество примеров применения этого метода суммирования можно найти в книге Г. П. Егорычева *Интегральное представление и вычисление комбинаторных сумм* (Новосибирск: Наука, 1977), 284 с. (G. P. Egorychev, *Integral Representation and the Computation of Combinatorial Sums* (Amer. Math. Soc., 1984)).

26. $[F(z)]\,G(z)$ обозначает постоянный член $F(z^{-1})G(z)$. Этот вопрос обсуждается в работе D. E. Knuth, *A Classical Mind* (Prentice-Hall, 1994), 247–258.

РАЗДЕЛ 1.2.10

1. $G_n(0) = 1/n$; это вероятность того, что $X[n]$ имеет наибольшее значение.

2. $G''(1) = \sum_k k(k-1)p_k$, $G'(1) = \sum_k k p_k$.

3. (min 0, ave 6.49, max 999, dev 2 42). Обратите внимание, что $H_n^{(2)}$ приближенно равно $\pi^2/6$; см. 1.2.7–(7).

4. $\binom{n}{k} p^k q^{n-k}$.

5. Среднее равно $36/5 = 7.2$, а среднее квадратичное отклонение — $6\sqrt{2}/5 \approx 1.697$.

6. Для (18) формула

$$\ln(q + pe^t) = \ln\Big(1 + pt + \frac{pt^2}{2} + \frac{pt^3}{6} + \cdots\Big) = pt + p(1-p)\frac{t^2}{2} + p(1-p)(1-2p)\frac{t^3}{6} + \cdots$$

говорит о том, что $\kappa_3/n = p(1-p)(1-2p) = pq(q-p)$. (Но эта схема не распространяется на коэффициент при t^4.) Полагая $p = k^{-1}$, получим $\kappa_3 = \sum_{k=2}^n k^{-1}(1-k^{-1})(1-2k^{-1}) = H_n - 3H_n^{(2)} + 2H_n^{(3)}$ для случая распределения (8). И для (20) имеем $\ln G(e^t) = t + H(nt) - H(t)$, где $H(t) = \ln((e^t-1)/t)$. Поскольку $H'(z) = e^t/(e^t-1) - 1/t$, в этом случае $\kappa_r = (n^r - 1)B_r/r$ для всех $r \ge 2$, в частности $\kappa_3 = 0$.

7. Вероятность того, что $A = k$, равна p_{mk}. Можно считать, что мы рассматриваем величины $1, 2, \ldots, m$. Для любого заданного разбиения n элементов на m непересекающихся множеств существует $m!$ способов присвоения этим множествам чисел $1, \ldots, m$. Алгоритм M работает с данными величинами так, как будто присутствуют только крайние справа элементы каждого множества Поэтому p_{mk} — среднее для любого фиксированного разбиения. Например, если $n = 5$, $m = 3$, то одно из разбиений выглядит так:

$$\{X[1], X[4]\} \quad \{X[2], X[5]\} \quad \{X[3]\};$$

возможными размещениями будут 12312, 13213, 21321, 23123, 31231, 32132. В каждом разбиении мы получаем одинаковое число размещений с $A = k$.

С другой стороны, если у нас больше информации, то распределение вероятностей изменится. Например, при $n = 3$ и $m = 2$ рассматриваются шесть возможностей: 122, 212, 221, 211, 121, 112 (см. рассуждения из предыдущего абзаца). Если же нам известно, что имеются две двойки и одна единица, то следует рассмотреть только первые три из приведенных шести возможностей. Но такая интерпретация не согласуется с постановкой задачи.

8. $M^{\underline{n}}/M^n$. Чем больше значение M, тем данная вероятность ближе к единице.

9. Пусть q_{nm} — вероятность того, что имеется ровно m различных значений; тогда из рекуррентного соотношения

$$q_{nm} = \frac{M-m+1}{M} q_{(n-1)(m-1)} + \frac{m}{M} q_{(n-1)m}$$

получим, что

$$q_{nm} = M! \left\{ {n \atop m} \right\} \Big/ (M-m)!\, M^n.$$

См. также упр. 1.2.6–64.

10. Нужно просуммировать $q_{nm} p_{mk}$ по всем m. Получим $M^{-n} \sum_m \binom{M}{m} \left\{ {n \atop m} \right\} \left[{m \atop k+1} \right]$. Формула для среднего, которое меньше, чем

$$H_M - \sum_{m=1}^{M} \left(1 - \frac{m}{M}\right)^n m^{-1} = H_n + \sum_{k=1}^{n} \left(\binom{n}{k} - 1 \right) B_k M^{-k} k^{-1},$$

далеко не проста.

11. Так как это произведение, семиинварианты складываются. Если $H(z) = z^n$, $H(e^t) = e^{nt}$, то $\kappa_1 = n$, а все остальные семиинварианты равны нулю. Следовательно, $\mathrm{mean}(F) = n + \mathrm{mean}(G)$, а все остальные семиинварианты остаются без изменений. (Этим объясняется название "семиинвариант"*.)

12. Первое тождество очевидно, если записать функцию e^{kt} в виде степенного ряда. Чтобы получить второе тождество, положим $u = 1 + M_1 t + M_2 t^2/2! + \cdots$. При $t = 0$ имеем $u = 1$ и $D_t^k u = M_k$. Кроме того, $D_u^j(\ln u) = (-1)^{j-1}(j-1)!/u^j$. Согласно упр. 11 такие же формулы применимы и для центральных моментов, если отбросить все члены, для которых $k_1 > 0$; таким образом, $\kappa_2 = m_2$, $\kappa_3 = m_3$, $\kappa_4 = m_4 - 3m_2^2$.

13. $G_n(z) = \dfrac{\Gamma(n+z)}{\Gamma(z+1)n!} = \dfrac{e^{-z}(n+z)^{z-1}}{\Gamma(z+1)} \left(1 + \dfrac{z}{n}\right)^n (1 + O(n^{-1})) = \dfrac{n^{z-1}}{\Gamma(z+1)} (1 + O(n^{-1}))$. Пусть $z_n = e^{it/\sigma_n}$. При $n \to \infty$ и фиксированном t имеем $z_n \to 1$. Следовательно, $\Gamma(z_n + 1) \to 1$ и

$$\lim_{n\to\infty} z_n^{-\mu_n} G_n(z_n) = \lim_{n\to\infty} \exp\left(\frac{-it\mu_n}{\sigma_n} + (e^{it/\sigma_n} - 1) \ln n \right)$$

$$= \lim_{n\to\infty} \exp\left(\frac{-t^2 \ln n}{2\sigma_n^2} + O\left(\frac{1}{\sqrt{\log n}} \right) \right) = e^{-t^2/2}.$$

Замечание. Это теорема Гончарова [*Изв. АН СССР. Сер. Мат.* **8** (1944), 3–48]. Ф. Флажоле (P. Flajolet) и М. Сориа (M. Soria) [*Disc. Math.* **114** (1993), 159–180] провели дополнительные исследования и показали, что распределение с производящей функцией $G_n(z)$ и большое семейство близких к нему распределений не только являются приближенно нормальными

* "Полунеизменный". — *Прим. перев.*

вблизи средних значений кроме того, хвосты данных распределений равномерно мажорируются экспонентой, т. е.

$$\text{вероятность}\left(\left|\frac{X_n-\mu_n}{\sigma_n}\right|>x\right)<e^{-ax}$$

для некоторой положительной константы a и для всех n и x.

14. $e^{-it pn/\sqrt{pqn}}(q+pe^{it/\sqrt{pqn}})^n = (qe^{-itp/\sqrt{pqn}}+pe^{itq/\sqrt{pqn}})^n$. Разложив экспоненциальные функции в степенные ряды, получим $(1-t^2/2n+O(n^{-3/2}))^n = \exp(n\ln(1-t^2/2n+O(n^{-3/2}))) = \exp(-t^2/2+O(n^{-1/2}))\to\exp(-t^2/2)$.

15. (a) $\sum_{k\ge0}e^{-\mu}(\mu z)^k/k! = e^{\mu(z-1)}$. (b) $\ln e^{\mu(e^t-1)} = \mu(e^t-1)$, поэтому все семиинварианты равны μ. (c) $\exp(-itnp/\sqrt{np})\,\exp(np(it/\sqrt{np}+-t^2/2np+O(n^{-3/2}))) = \exp(-t^2/2+O(n^{-1/2}))$.

16. $g(z)=\sum_k p_k g_k(z)$, $\quad\text{mean}(g)=\sum_k p_k\,\text{mean}(g_k)\quad$ и $\quad\text{var}(g)=\sum_k p_k\,\text{var}(g_k)+\sum_{j<k}p_jp_k(\text{mean}(g_j)-\text{mean}(g_k))^2$.

17. (a) Коэффициенты разложений $f(z)$ и $g(z)$ неотрицательны и $f(1)=g(1)=1$. Очевидно, что $h(z)$ имеет такие же свойства, поскольку $h(1)=g(f(1))$, а коэффициенты разложения h являются многочленами от неотрицательных коэффициентов f и g. (b) Пусть $f(z)=\sum p_kz^k$; p_k — это вероятность того, что случайная величина X_f, соответствующая $f(z)$, принимает значение k, $p_k = PX_f=k$. Пусть $g(z)=\sum q_kz^k$; q_k — вероятность того, что случайная величина X_g, соответствующая $g(z)$, принимает значение k. Тогда $h(z)=\sum r_kz^k$ соответствует случайной величине X_h такого вида: $X_h = \sum k=1_g^X X_{fk}$, где X_{fk} — независимые одинаково распределенные случайные величины, каждая из которых распределена так же, как X_f, и все они не зависят от X_g. (Это легко показать, если заметить, что $f(z)^k=\sum s_tz^t$, где s_t — вероятность того, что сумма k независимых случайных величин X_{fj} равна t.) *Пример.* Если f дает вероятности того, что мужчина имеет k потомков мужского пола, а g — вероятности того, что в n-м поколении k мужчин, то h порождает вероятности того, что в $(n+1)$-м поколении будет k мужчин (при условии независимости). (c) $\text{mean}(h)=\text{mean}(g)\,\text{mean}(f)$; $\text{var}(h)=\text{var}(g)\,\text{mean}^2(f)+\text{mean}(g)\,\text{var}(f)$.

18. Рассмотрим выбор величин $X[1],\dots,X[n]$ как процесс, в котором мы сначала размещаем все числа n, затем среди них размещаем все $(n-1),\dots$ и наконец среди всего остального размещаем единицы. Когда мы размещаем числа r среди чисел $r+1,\dots,n$, число локальных максимумов при движении справа налево возрастает на единицу тогда и только тогда, когда мы помещаем число r в крайнюю позицию справа. Вероятность этого события равна $k_r/(k_r+k_{r+1}+\cdots+k_n)$.

19. Положим $a_k=l$. Тогда a_k — максимум слева направо последовательности $a_1\dots a_n$ $\iff$ $j<k$ влечет $a_j<l$ $\iff$ $a_j>l$ влечет $j>k$ $\iff$ $j>l$ влечет $b_j>k$ $\iff$ k — минимум справа налево последовательности $b_1\dots b_n$.

20. Имеем $m_L=\max\{a_1-b_1,\dots,a_n-b_n\}$. Доказательство. Предположим, что это не так. Пусть k — наименьший индекс, для которого $a_k-b_k>m_L$. Тогда a_k не является максимумом слева направо и, значит, существует такое $j<k$, что $a_j\ge a_k$. Но тогда $a_j-b_j\ge a_k-b_k>m_L$, что противоречит минимальности k. Аналогично $m_R=\max\{b_1-a_1,\dots,b_n-a_n\}$.

21. При $\epsilon\ge q$ результат тривиален, поэтому предположим, что $\epsilon<q$. Полагая $x=\frac{p+\epsilon}{p}\frac{q}{q-\epsilon}$ в (25), получим $\Pr(X\ge n(p+\epsilon))\le((\frac{p}{p+\epsilon})^{p+\epsilon}(\frac{q}{q-\epsilon})^{q-\epsilon})^n$. Имеем $(\frac{p}{p+\epsilon})^{p+\epsilon}\le e^{-\epsilon}$, так как $t\le e^{t-1}$ для всех действительных t. И $(q-\epsilon)\ln\frac{q}{q-\epsilon}=\epsilon-\frac{1}{2\,1}\epsilon^2q^{-1}-\frac{1}{3\,2}\epsilon^3q^{-2}-\cdots\le\epsilon-\frac{1}{2q}\epsilon^2$. (Более подробный анализ дает несколько более сильную оценку $\exp(-\epsilon^2n/(2pq))$, где $p\ge\frac12$; далее легко получить верхнюю грань $\exp(-2\epsilon^2n)$ для всех p.)

Поменяв местами роли "орла" и "решки", получим

$$\Pr(X\le n(p-\epsilon))=\Pr(n-X\ge n(q+\epsilon))\le e^{-\epsilon^2n/(2p)}.$$

22. (a) В (24) и (25) положим $x = r$ и заметим, что $q_k + p_k r = 1 + (r-1)p_k \le e^{(r-1)p_k}$. [См. H. Chernoff, *Annals of Math. Stat.* **23** (1952), 493–507.]

(b) Положим $r = 1 + \delta\sqrt{}$, где $|\delta| \le 1$. Тогда $r^{-r}e^{r-1} = \exp(-\frac{1}{2\cdot1}\delta^2 + \frac{1}{3\cdot2}\delta^3 - \cdots)$. Это выражение $\le e^{-\delta^2/2}$ при $\delta \le 0$ и $\le e^{-\delta^2/3}$ при $\delta \ge 0$

(c) По мере того как r возрастает от 1 до ∞, функция $r^{-1}e^{1-r^{-1}}$ убывает от 1 до 0. Если $r \ge 2$, то значение функции $\le \frac{1}{2}e^{1/2} < .825$; если же $r \ge 4.32$, то значение функции $< \frac{1}{2}$.

Интересно заметить, что неравенства для хвостов распределений при $x = r$ дают точно такую же оценку $(r^{-r}e^{r-1})^\mu$, если X из упр. 15 имеет распределение Пуассона.

23. Полагая в (24) $x = \frac{p-\epsilon}{p}\frac{q}{q-\epsilon}$, получим $\Pr(X \le n(p-\epsilon)) \le ((\frac{p}{p-\epsilon})^{p-\epsilon}(\frac{q-\epsilon}{q})^{q-\epsilon})^n \le e^{-\epsilon^2 n/(2pq)}$. Аналогично при $x = \frac{p+\epsilon}{p}\frac{q}{q+\epsilon}$ получим $\Pr(X \ge n(p+\epsilon)) \le ((\frac{p}{p+\epsilon})^{p+\epsilon}(\frac{q+\epsilon}{q})^{q+\epsilon})^n$. Положим $f(\epsilon) = (q+\epsilon)\ln(1+\frac{\epsilon}{q}) - (p+\epsilon)\ln(1+\frac{\epsilon}{p})$ и заметим, что $f'(\epsilon) = \ln(1+\frac{\epsilon}{q}) - \ln(1+\frac{\epsilon}{p})$. Отсюда следует, что $f(\epsilon) \le -\epsilon^2/(6pq)$, если $0 \le \epsilon \le p$.

РАЗДЕЛ 1.2.11.1

1. Нулю.

2. Символы O могут представлять различные приближенные величины. Поскольку левой частью может быть $f(n) - (-f(n)) = 2f(n)$, самое большее, что можно сказать, — это $O(f(n)) - O(f(n)) = O(f(n))$ (следует из (6) и (7)). Чтобы доказать (7), нужно заметить, что если $|x_n| \le M|f(n)|$ для $n \ge n_0$ и $|x'_n| \le M'|f(n)|$ для $n \ge n'_0$, то $|x_n \pm x'_n| \le |x_n| + |x'_n| \le (M+M')|f(n)|$ для $n \ge \max(n_0, n'_0)$. (Подпись: Студент Квик*.)

3. $n(\ln n) + \gamma n + O(\sqrt{n}\ln n)$.

4. $\ln a + (\ln a)^2/2n + (\ln a)^3/6n^2 + O(n^{-3})$.

5. Если $f(n) = n^2$ и $g(n) = 1$, то n принадлежит множеству $O(f(n) + g(n))$, но не множеству $f(n) + O(g(n))$. Поэтому утверждение ложно.

6. Символов O переменное число, а именно — n. Их заменили единственным символом O, ошибочно полагая, что одного значения M будет достаточно для каждого неравенства $|kn| \le Mn$. Как мы знаем, на самом деле данная сумма равна $\Theta(n^3)$. Последнее равенство, $\sum_{k=1}^n O(n) = O(n^2)$, совершенно справедливо.

7. Если рассмотреть степенной ряд 1.2.9–(22), то видно, что для положительных x выполняется неравенство $e^x > x^{m+1}/(m+1)!$ Отсюда следует, что отношение e^x/x^m нельзя ограничить ни одним M.

8. Сделаем замену n на e^n и применим метод из предыдущего упражнения.

9. Если $|f(x)| \le M|z|^m$ для $|z| \le r$, то $e^{f(z)} \le e^{M|z|^m} = 1 + |z|^m(M + M^2|z|^m/2! + M^3|z|^{2m}/3! + \cdots) \le 1 + |z|^m(M + M^2r^m/2! + M^3r^{2m}/3! + \cdots)$.

10. $\ln(1+O(z^m)) = O(z^m)$, если m — положительное целое. *Доказательство.* Если $f(z) = O(z^m)$, то существуют положительные числа $r < 1$, $r' < 1$ и константа M, такие, что $|f(z)| \le M|z|^m \le r'$ при $|z| \le r$. Тогда $|\ln(1+f(z))| \le |f(z)| + \frac{1}{2}|f(z)|^2 + \cdots \le |z|^m M(1 + \frac{1}{2}r' + \cdots)$.

11. Формулу (12) можно применить для $m = 1$ и $z = \ln n/n$. Это справедливо, поскольку $\ln n/n \le r$ для любого заданного $r > 0$, если n достаточно велико.

12. Пусть $f(z) = (ze^z/(e^z-1))^{1/2}$. Если бы $\left[{1/2 \atop 1/2-k}\right]$ было равно $O(n^k)$, то из вспомогательного тождества следовало бы, что $[z^k]\,f(z) = O(n^k/(k-1)^t)$, поэтому степенной ряд для $f(z)$ сходился бы при $z = 2\pi\imath$. Но $f(2\pi\imath) = \infty$.

13. В определениях O и Ω можно взять $L = 1/M$.

---

* В оригинале — J. H. Quick. От англ. "quick" (здесь — "сообразительный"). — *Прим. перев.*

РАЗДЕЛ 1.2.11.2

1. $(B_0+B_1z+B_2z^2/2!+\cdots)e^z=(B_0+B_1z+B_2z^2/2!+\cdots)+z$; примените формулу 1.2.9–(11).

2. Для того чтобы можно было выполнить интегрирование по частям, функция $B_{m+1}(\{x\})$ должна быть непрерывной.

3. $|R_{mn}| \le |B_m/(m)!|\int_1^n |f^{(m)}(x)|\,dx$. [*Замечания.* $B_m(x) = (-1)^m B_m(1-x)$ и $B_m(x)$ равно $m!$, умноженному на коэффициент при z^m в разложении $ze^{xz}/(e^z-1)$. В частности, так как $e^{z/2}/(e^z-1) = 1/(e^{z/2}-1) - 1/(e^z-1)$, имеем $B_m(\frac12) = (2^{1-m}-1)B_m$. Нетрудно доказать, что максимум $|B_m - B_m(x)|$ для $0 \le x \le 1$ достигается при $x = \frac12$, когда m четно. А теперь при $m = 2k \ge 4$ давайте просто запишем R_m и C_m для величин R_{mn} и C_{mn}. Имеем $R_{m-2} = C_m + R_m = \int_1^n (B_m - B_m(\{x\}))f^{(m)}(x)\,dx/m!$ и $B_m - B_m(\{x\})$ лежит между 0 и $(2-2^{1-m})B_m$. Следовательно, R_{m-2} лежит между 0 и $(2-2^{1-m})C_m$. Отсюда получаем, что R_m лежит между $-C_m$ и $(1-2^{1-m})C_m$ (это более сильный результат). Таким образом, видно, что если $f^{(m+2)}(x)\,f^{(m+4)}(x) > 0$ для $1 < x < n$, то величины C_{m+2} и C_{m+4} имеют противоположные знаки, в то время как R_m имеет такой же знак, как C_{m+2}, R_{m+2} имеет такой же знак, как C_{m+4}, и $|R_{m+2}| \le |C_{m+2}|$; это доказывает формулу (13). [См. J. F. Steffensen, *Interpolation* (Baltimore, 1937), §14.]

4. $$\sum_{0\le k<n} k^m = \frac{n^{m+1}}{1+m} + \sum_{k=1}^{m} \frac{B_k}{k!}\frac{m!}{(m-k+1)!}n^{m-k+1} = \frac{1}{m+1}B_{m+1}(n) - \frac{1}{m+1}B_{m+1}.$$

5.
$$\kappa = \sqrt{2}\lim_{n\to\infty}\frac{2^{2n}(n!)^2}{\sqrt{n}\,(2n)!};$$
$$\kappa^2 = \lim_{n\to\infty}\frac{2}{n}\,\frac{n^2(n-1)^2\ldots(1)^2}{(n-\frac12)^2(n-\frac32)^2\ldots(\frac12)^2} = 4\,\frac{2\cdot2\cdot4\cdot4\cdot\cdots}{1\cdot3\cdot3\cdot5\cdot\cdots} = 2\pi.$$

6. Пусть $c > 0$. Рассмотрим сумму $\sum_{0\le k<n}\ln(k+c)$. Находим

$$\ln(c(c+1)\ldots(c+n-1)) = (n+c)\ln(n+c) - c\ln c - n - \tfrac12\ln(n+c) + \tfrac12\ln c + \sum_{1<k\le m}\frac{B_k(-1)^k}{k(k-1)}\left(\frac{1}{(n+c)^{k-1}} - \frac{1}{c^{k-1}}\right) + R_{mn}.$$

Кроме того,

$$\ln(n-1)! = (n-\tfrac12)\ln n - n + \sigma + \sum_{1<k\le m}\frac{B_k(-1)^k}{k(k-1)}\left(\frac{1}{n^{k-1}}\right) - \frac1m\int_n^\infty \frac{B_m(\{x\})\,dx}{x^m}.$$

Но $\ln\Gamma_{n-1}(c) = c\ln(n-1) + \ln(n-1)! - \ln(c\ldots(c+n-1))$. Выполняя подстановку, при $n\to\infty$ получаем

$$\ln\Gamma(c) = -c + (c-\tfrac12)\ln c + \sigma + \sum_{1<k\le m}\frac{B_k(-1)^k}{k(k-1)c^{k-1}} - \frac1m\int_0^\infty \frac{B_m(\{x\})\,dx}{(x+c)^m}.$$

Отсюда следует, что $\Gamma(c+1) = ce^{\ln\Gamma(c)}$ имеет такое же асимптотическое представление, как и $c!$.

7. $An^{n^2/2+n/2+1/12}e^{-n^2/4}$, где A — константа. Для получения этого результата примените формулу суммирования Эйлера к $\sum_{k=1}^{n-1} k\ln k$. Чтобы получить более точную формулу, нужно умножить полученный результат на

$$\exp(-B_4/(2\cdot3\cdot4n^2) - \cdots - B_{2t}/((2t-2)(2t-1)(2t)n^{2t-2}) + O(1/n^{2t})).$$

Число A называется постоянной Глейшера и равно 1.2824271... [*Messenger of Math.* **7** (1877), 43–47]. Эта постоянная, как можно показать, равна

$$e^{1/12-\zeta'(-1)} = (2\pi e^{\gamma-\zeta'(2)/\zeta(2)})^{1/12}$$

[de Bruijn, *Asymptotic Methods in Analysis*, §3.7].

8. Мы имеем, например, $\ln(an^2+bn) = 2\ln n + \ln a + \ln(1+b/(an))$. Поэтому ответом на первый вопрос будет $2an^2\ln n + a(\ln a - 1)n^2 + 2bn\ln n + bn\ln a + \ln n + b^2/(2a) + \sigma + (3a-b^2)b/(6a^2n) + O(n^{-2})$. При вычислении величины $\ln(cn^2)! - \ln(cn^2-n)! - n\ln c - \ln n^2! + \ln(n^2-n)! = (c-1)/(2c) - (c-1)(2c-1)/(6c^2n) + O(n^{-2})$ после многочисленных сокращений получим

$$e^{(c-1)/(2c)}\Big(1 - \frac{(c-1)(2c-1)}{6c^2n}\Big)(1+O(n^{-2})).$$

Кстати, $\binom{cn^2}{n}/c^n\binom{n^2}{n}$ можно записать в виде $\prod_{j=1}^{n-1}(1+\alpha j/(n^2-j))$, где $\alpha = 1-1/c$.

9. (a) Имеем $\ln(2n)! = (2n+\frac12)\ln 2n - 2n + \sigma + \frac{1}{24n} + O(n^{-3})$ и $\ln(n!)^2 = (2n+1)\ln n - 2n + 2\sigma + \frac{1}{6n} + O(n^{-3})$. Следовательно, $\binom{2n}{n} = \exp(2n\ln 2 - \frac12\ln\pi n - \frac{1}{8n} + O(n^{-3})) = 2^{2n}(\pi n)^{-1/2}(1 - \frac18 n^{-1} + \frac{1}{128}n^{-2} + O(n^{-3}))$. (b) Поскольку $\binom{2n}{n} = 2^{2n}\binom{n-1/2}{n}$ и $\binom{n-1/2}{n} = \Gamma(n+1/2)/(n\Gamma(n)\Gamma(1/2)) = n^{-1}n^{1/2}/\sqrt{\pi}$, из 1.2.11.1–(16) получаем такой же результат, потому что

$$\begin{bmatrix}1/2\\1/2\end{bmatrix} = 1, \quad \begin{bmatrix}1/2\\-1/2\end{bmatrix} = \binom{1/2}{2} = -\frac18, \quad \begin{bmatrix}1/2\\-3/2\end{bmatrix} = \binom{1/2}{4} + 2\binom{3/2}{4} = \frac{1}{128}.$$

Метод (b) объясняет, почему все знаменатели в

$$\binom{2n}{n} = \frac{2^{2n}}{\sqrt{\pi n}}\Big(1 - \frac{n^{-1}}{8} + \frac{n^{-2}}{128} + \frac{5n^{-3}}{1024} - \frac{21n^{-4}}{32768} - \frac{399n^{-5}}{262144} + \frac{869n^{-6}}{4194304} + O(n^{-7})\Big)$$

имеют степень 2 [Knuth, Vardi, *AMM* **97** (1990), 629–630].

РАЗДЕЛ 1.2.11.3

1. Проинтегрируйте по частям.

2. Подставляем в интеграл ряд для e^{-t}.

3. См. формулу 1.2.9–(11) и упр. 1.2.6–48.

4. $1 + 1/u$ ограничена как функция от v, поскольку она стремится к нулю, когда v пробегает промежуток от r до бесконечности. Замените $1+1/u$ на M, и тогда полученный интеграл будет равен Me^{-rx}.

5. Функция $f''(x) = f(x)((n+1/2)(n-1/2)/x^2 - (2n+1)/x + 1)$ меняет знак в точке $r = n + 1/2 - \sqrt{n+1/2}$, поэтому $|R| = O(\int_0^n |f''(x)|\,dx) = O(\int_0^r f''(x)\,dx - \int_r^n f''(x)\,dx) = O(f'(n) - 2f'(r) + f'(0)) = O(f(n)/\sqrt{n})$.

6. Приведем левую часть к виду $n^{n+\beta}\exp((n+\beta)(\alpha/n - \alpha^2/2n^2 + O(n^{-3})))$ и т. д.

7. В подынтегральном выражении, представленном в виде ряда по степеням x^{-1}, коэффициент при x^{-n} имеет порядок $O(u^{2n})$. После интегрирования члены с x^{-3} будут иметь вид $Cu^7/x^3 = O(x^{-5/4})$ и т. д. Чтобы получить в ответе точность порядка $O(x^{-2})$, можно отбросить члены u^n/x^m с $4m-n \ge 9$. Тогда, разлагая произведение $\exp(-u^2/2x)\exp(u^3/3x^2)\ldots$, в конце концов получим ответ

$$yx^{1/4} - \frac{y^3}{6}x^{-1/4} + \frac{y^5}{40}x^{-3/4} + \frac{y^4}{12}x^{-1} - \frac{y^7}{336}x^{-5/4} - \frac{y^6}{36}x^{-3/2} + \Big(\frac{y^9}{3456} - \frac{y^5}{20}\Big)x^{-7/4} + O(x^{-2}).$$

8. (Решение Миклоша Шимоновица (Miklós Simonovits).) Для достаточно больших x имеем $|f(x)| < x$. Обозначим через $R(x) = \int_0^{f(x)} (e^{-g(u,x)} - e^{-h(u,x)})\,du$ разность между двумя заданными интегралами, где

$$g(u,x) = u - x\ln(1+u/x) \quad \text{и} \quad h(u,x) = u^2/2x - u^3/3x^2 + \cdots + (-1)^m u^m/mx^{m-1}.$$

Заметьте, что $g(u,x) \geq 0$ и $h(u,x) \geq 0$ при $|u| < x$; кроме того,

$$g(u,x) = h(u,x) + O(u^{m+1}/x^m).$$

По теореме о среднем значении $e^a - e^b = (a-b)\,e^c$ для некоторого c, лежащего между a и b. Поэтому $|e^a - e^b| \leq |a-b|$, если $a, b \leq 0$. Отсюда следует, что

$$|R(x)| \leq \int_{-|f(x)|}^{|f(x)|} |g(u,x) - h(u,x)|\,du = O\left(\int_{-Mx^r}^{Mx^r} \frac{u^{m+1}\,du}{x^m}\right)$$
$$= O(x^{(m+2)r-m}) = O(x^{-s}).$$

9. Можем предполагать, что $p \neq 1$, так как случай $p = 1$ описывается в теореме A. Будем считать также, что $p \neq 0$, так как случай $p = 0$ тривиален.

Случай 1: $p < 1$. Сделаем подстановку $t = px(1-u)$ и обозначим $v = -\ln(1-u) - pu$. Так как $dv = ((1-p+pu)/(1-u))\,du$, функция v монотонна при $0 \leq u \leq 1$ и мы получаем интеграл вида

$$\int_0^\infty xe^{-xv}dv\Big(\frac{1-u}{1-p+pu}\Big).$$

Поскольку выражение в скобках равно $(1-p)^{-1}(1 - v(1-p)^{-2} + \cdots)$, получаем ответ

$$\frac{p}{1-p}(pe^{1-p})^x \frac{e^{-x}x^z}{\Gamma(x+1)}\left(1 - \frac{1}{(p-1)^2x} + O(x^{-2})\right).$$

Случай 2: $p > 1$. В этом случае имеем $1 - \int_{px}^\infty(\ )$. В последнем интеграле делаем подстановку $t = px(1+u)$, затем обозначаем $v = pu - \ln(1+u)$ и продолжаем рассуждения, как в случае 1. В ответе получаем точно такую же формулу, как в случае 1, плюс единица. Заметьте, что $pe^{1-p} < 1$, поэтому $(pe^{1-p})^x$ *очень* мало.

Ответ к упр. 11 дает еще один способ решения данной задачи.

10. $\dfrac{p}{p-1}(pe^{1-p})^x e^{-x}x^x\left(1 - e^{-y} - \dfrac{e^{-y}(e^y - 1 - y - y^2/2)}{x(p-1)^2} + O(x^{-2})\right).$

11. Прежде всего, получим равенство $xQ_x(n) + R_{1/x}(n) = n!\,(x/n)^n e^{n/x}$, которое является обобщением (4). Кроме того, получим $R_x(n) = n!\,(e^x/nx)^n\gamma(n,nx)/(n-1)!$, что является обобщением (9). Поскольку $a\gamma(a,x) = \gamma(a+1,\,x) + e^{-x}x^a$, можно записать также $R_x(n) = 1 + (e^x/nx)^n\gamma(n+1,\,nx)$, тем самым связав эту задачу с упр. 9. Более того, можно непосредственно заняться функциями $Q_x(n)$ и $R_x(n)$, воспользовавшись соотношениями 1.2.9–(27) и (28), чтобы получить разложения в ряд, включающие числа Стирлинга:

$$1 + xQ_x(n) = \sum_{k\geq 0} x^k n^{\underline{k}}/n^k = \sum_{k,m} \frac{(-1)^m}{n^m}\begin{bmatrix} k \\ k-m \end{bmatrix} x^k;$$

$$R_x(n) = \sum_{k\geq 0} x^k n^{\underline{k}}/(n+1)^{\bar{k}} = \sum_{k,m} \frac{(-1)^m}{n^m}\begin{Bmatrix} k+m \\ k \end{Bmatrix} x^k.$$

Эти суммы по k сходятся для фиксированного m при $|x| < 1$, а при $|x| > 1$ можно воспользоваться соотношением между $Q_x(n)$ и $R_{1/x}(n)$. В результате получим формулы

$$Q_x(n) = \frac{1}{1-x} - \frac{x}{(1-x)^3 n} + \cdots + \frac{(-1)^m q_m(x)}{(1-x)^{2m+1} n^m} + O(n^{-1-m}),$$
$$R_x(n) = \frac{1}{1-x} - \frac{x}{(1-x)^3 n} + \cdots + \frac{(-1)^m r_m(x)}{(1-x)^{2m+1} n^m} + O(n^{-1-m}), \quad \text{если } x < 1;$$
$$Q_x(n) = \frac{n!\, x^{n-1} e^{n/x}}{n^n} + \frac{1}{1-x} - \frac{x}{(1-x)^3 n} + \cdots + \frac{(-1)^m q_m(x)}{(1-x)^{2m+1} n^m} + O(n^{-1-m}),$$
$$R_x(n) = \frac{n!\, e^{nx}}{n^n x^n} + \frac{1}{1-x} - \frac{x}{(1-x)^3 n} + \cdots + \frac{(-1)^m r_m(x)}{(1-x)^{2m+1} n^m} + O(n^{-1-m}), \quad \text{если } x > 1.$$

Здесь

$$q_m(x) = \left\langle\!\left\langle {m \atop 0} \right\rangle\!\right\rangle x^{2m-1} + \left\langle\!\left\langle {m \atop 1} \right\rangle\!\right\rangle x^{2m-2} + \cdots$$

и

$$r_m(x) = \left\langle\!\left\langle {m \atop 0} \right\rangle\!\right\rangle x + \left\langle\!\left\langle {m \atop 1} \right\rangle\!\right\rangle x^2 + \cdots$$

являются многочленами, коэффициенты которых — эйлеровы числа второго порядка [*CMath* §6.2; см. L. Carlitz, *Proc. Amer. Math. Soc.* **16** (1965), 248–252]. Случай $x = -1$ является несколько более сложным, но здесь можно воспользоваться непрерывностью, так как постоянные из определения $O(n^{-1-m})$ не зависят от x при $x < 0$. Интересно отметить, что разность $R_{-1}(n) - Q_{-1}(n) = (-1)^n n!/e^n n^n \approx (-1)^n \sqrt{2\pi n}/e^{2n}$ крайне мала.

12. $\gamma(\frac{1}{2}, \frac{1}{2}x^2)/\sqrt{2}$.

13. См. P. Flajolet, P. Grabner, P. Kirschenhofer, H. Prodinger, *J. Computational and Applied Math.* **58** (1995), 103–116.

15. Раскрывая подынтегральное выражение с помощью биномиальной теоремы, получим $1 + Q(n)$.

16. Запишем $Q(k)$ в виде суммы и изменим порядок суммирования с помощью соотношения 1.2.6–(53).

17. $S(n) = \sqrt{\pi n/2} + \frac{2}{3} - \frac{1}{24}\sqrt{\pi/2n} - \frac{4}{135}n^{-1} + \frac{49}{1152}\sqrt{\pi/2n^3} + O(n^{-2})$. [Заметьте, что $S(n+1) + P(n) = \sum_{k\ge 0} k^{n-k} k!/n!$, в то время как $Q(n) + R(n) = \sum_{k\ge 0} n!/k!\, n^{n-k}$.]

18. Пусть $S_n(x,y) = \sum_k \binom{n}{k}(x+k)^k (y+n-k)^{n-k}$. Тогда для $n > 0$ имеем

$$\begin{aligned} S_n(x,y) &= x\textstyle\sum_k \binom{n}{k}(x+k)^{k-1}(y+n-k)^{n-k} + n\sum_k \binom{n-1}{k}(x+1+k)^k (y+n-1-k)^{n-1-k} \\ &= (x+y+n)^n + nS_{n-1}(x+1,y) \end{aligned}$$

по формуле Абеля 1.2.6–(16). Следовательно,

$$S_n(x,y) = \textstyle\sum_k \binom{n}{k} k!\,(x+y+n)^{n-k}.$$

[Эта формула принадлежит Коши, который доказал ее с помощью вычетов; см. его работу *Œuvres* (2) **6**, 62–73.] Значит, исходные суммы равны $n^n(1 + Q(n))$ и $(n+1)^n Q(n+1)$ соответственно.

19. Предположим, C_n существует для всех $n \ge N$ и $|f(x)| \le Mx^\alpha$, где $0 \le x \le r$. Обозначим $F(x) = \int_r^x e^{-Nt} f(t)\,dt$. Тогда при $n > N$ имеем

$$\begin{aligned}|C_n| &\le \int_0^r e^{-nx}|f(x)|\,dx \;+\; \left|\int_r^\infty e^{-(n-N)x} e^{-Nx} f(x)\,dx\right| \\ &\le M\int_0^r e^{-nx}x^\alpha\,dx \;+\; (n-N)\left|\int_r^\infty e^{-(n-N)x}F(x)\,dx\right| \\ &\le M\int_0^\infty e^{-nx}x^\alpha\,dx \;+\; (n-N)\sup_{x\ge r}|F(x)|\int_r^\infty e^{-(n-N)x}\,dx \\ &= M\Gamma(\alpha+1)n^{-1-\alpha} + \sup_{x\ge r}|F(x)|e^{-(n-N)r} = O(n^{-1-\alpha}).\end{aligned}$$

[E. W. Barnes, *Phil. Trans.* **A206** (1906), 249–297; G. N. Watson, *Proc. London Math. Soc.* **17** (1918), 116–148.]

20. [C. C. Rousseau, *Applied Math. Letters* **2** (1989), 159–161.] Имеем

$$Q(n)+1 = n\int_0^\infty e^{-nx}(1+x)^n\,dx = n\int_0^\infty e^{-n(x-\ln(1+x))}\,dx = n\int_0^\infty e^{-nu}g(u)\,du,$$

подставляя $u = x - \ln(1+x)$ и полагая $g(u) = dx/du$. Заметим, что $x = \sum_{k=1}^\infty c_k(2u)^{k/2}$ для достаточно малых u. Отсюда $g(u) = \sum_{k=1}^{m-1} c_k(2u)^{k/2-1} + O(u^{m/2-1})$ и можно применить лемму Ватсона к $Q(n)+1-n\int_0^\infty e^{-nu}\sum_{k=1}^{m-1} kc_k(2u)^{k/2-1}\,du$.

РАЗДЕЛ 1.3.1

1. Четыре; тогда байт мог бы содержать $3^4 = 81$ различных значений.

2. Пять, так как пяти байтов хватило бы в любом случае, а четырех — нет.

3. (0:2); (3:3); (4:4); (5:5).

4. Предположительно индексный регистр 4 содержит значение, большее или равное 2 000, поэтому после индексирования получится допустимый адрес памяти.

5. "`DIV -80,3(0:5)`" или просто "`DIV -80,3`".

6. (a) rA ← | - | 5 | 1 | 200 | 15 |. (b) rI2 ← −200. (c) rX ← | + | 0 | 0 | 5 | 1 | ? |. (d) Не определена; нельзя загрузить такое большое значение в индексный регистр. (e) rX ← | - | 0 | 0 | 0 | 0 | 0 |.

7. Пусть $n = |\mathrm{rAX}|$ — это разрядность регистров A и X до операции, а $d = |V|$ — разрядность делителя. После операции разрядность rA равна $\lfloor n/d \rfloor$, а разрядность rX равна $n \bmod d$. Знак rX после операции будет таким же, как и предыдущий знак rA. Знаком rA после операции будет "+", если до операции знаки rA и V были одинаковы, и "−" — в противном случае.

Сформулируем это иначе. Если знаки rA и V одинаковы, то rA ← $\lfloor \mathrm{rAX}/V \rfloor$ и rX ← rAX mod V. В противном случае rA ← $\lceil \mathrm{rAX}/V \rceil$ и rX ← rAX mod −V.

8. rA ← | + | 0 | 617 | 0 | 1 |; rX ← | - | 0 | 0 | 0 | 1 | 1 |.

9. `ADD`, `SUB`, `DIV`, `NUM`, `JOV`, `JNOV`, `INCA`, `DECA`, `INCX`, `DECX`.

10. `CMPA`, `CMP1`, `CMP2`, `CMP3`, `CMP4`, `CMP5`, `CMP6`, `CMPX`. (А также `FCMP`, если рассматривать операции с плавающей точкой.)

11. `MOVE`, `LD1`, `LD1N`, `INC1`, `DEC1`, `ENT1`, `ENN1`.

12. INC3 0,3.

13. При замене на "JOV 1000" разница будет только во времени выполнения. Команда "JNOV 1001" в большинстве случаев изменяет содержимое rJ. При замене на "JNOV 1000" разница очень велика, так как эта команда может ввести компьютер в состояние выполнения бесконечного цикла.

14. Для NOP все ясно. ADD, SUB с F = (0:0) или если в поле адреса стоит "*" (вместо которой подставляется адрес команды) и F = (3:3); HLT (в зависимости от того, как вы интерпретируете формулировку упражнения); любые команды сдвига, поле адреса и поле индекса которых равны нулю; SLC или SRC с индексом, равным 0, и адресом, кратным 10; MOVE с F = 0; JSJ *+1; все команды INC или DEC с адресом и индексом, равным нулю. Но "ENT1 0,1" не всегда можно сделать эквивалентной NOP, так как она может изменить содержимое rI1 с -0 на $+0$.

15. 70; 80; 120. (Размер блока, умноженный на 5.)

16. (a) STZ 0; ENT1 1; MOVE 0(49); MOVE 0(50). Если бы было известно, что размер байта равен 100, то понадобилась бы только одна команда MOVE. Но нам не разрешено делать какие-либо предположения о размере байта. (b) Используйте 100 команд STZ.

17. (a) STZ 0,2; DEC2 1; J2NN 3000.

```
(b)     STZ  0
        ENT1 1
        JMP  3004
(3003)  MOVE 0(63)
(3004)  DEC2 63
        J2P  3003
        INC2 63
        ST2  3008(4:4)
(3008)  MOVE 0
```

(В немного более быстрой, но совершенно абсурдной программе используется 993 команды STZ: JMP 3995; STZ 1,2; STZ 2,2; ...; STZ 993,2; J2N 3999; DEC2 993; J2NN 3001; ENN1 0,2; JMP 3000,1.)

18. (Если правильно выполнить все команды, то на команде ADD произойдет переполнение, после чего в регистре A окажется нуль со знаком "$-$".) *Ответ.* Значение флага переполнения — единица, флага сравнения — EQUAL, содержимое rA — | - | 30 | 30 | 30 | 30 | 30 |, rX — | - | 31 | 30 | 30 | 30 | 30 |, rI1 — $+3$, а ячеек памяти 0001, 0002 — $+0$ (если только сама программа не начинается в ячейке 0000).

19. $42u = (2+1+2+2+1+1+1+2+2+1+2+2+3+10+10)u$.

20. (X. Фукуока (H. Fukuoka).)

```
(3991)  ENT1 0
        MOVE 3995    (Стандартное значение F для MOVE равно 1.)
(3993)  MOVE 0(43)   (3999 = 93 умножить на 43.)
        JMP  3993
(3995)  HLT  0
```

21. (a) Нет, если только не занести в него нуль с помощью внешних средств (см. о кнопке "GO" в упр. 26), так как программа может присвоить rJ $\leftarrow N$ только в результате перехода от ячейки $N-1$.

```
(b)     LDA  -1,4
        LDX  3004
```

```
        STX  -1,4
        JMP  -1,4
(3004)  JMP  3005
(3005)  STA  -1,4
```

22. *Минимальное время.* Если b — размер байта, то согласно нашему предположению $|X^{13}| < b^5$ и, следовательно, $X^2 < b$, поэтому X^2 может содержаться в одном байте. Данный факт положен в основу оригинального решения, предложенного Й. Н. Пэттом (Y. N Patt). Знаком rA будет знак X.

```
(3000)  LDA  2000
        MUL  2000(1:5)
        STX  3500(1:1)
        SRC  1
        MUL  3500
        STA  3501
        ADD  2000
        MUL  3501(1:5)
        STX  3501
        MUL  3501(1:5)
        SLAX 1
        HLT  0
(3500)  NOP  0
(3501)  NOP  0
```

rA					rX				
X^2	0	0	0	0	0	0	0	0	0
X^4		0	0	0	0	0	0	0	0
X^4		0	0	0	0	0	0	0	0
X^4		0	0	X	0	0	0	0	0
X^8				0	X^5		0	0	0
X^8				0	X^5		0	0	0
0	X^{13}					0	0	0	0
X^{13}					0	0	0	0	0

Занимаемый объем памяти — 14; время выполнения — $54u$ без учета команды HLT.

Согласно теории, изложенной в разделе 4.6.3, в данной ситуации "необходимо" выполнить по меньшей мере пять операций умножения, а в этой программе их всего четыре! Но на самом деле существует еще более удачное решение, которое приведено ниже.

Минимальный объем.

```
(3000)  ENT4 12          DEC4 1
        LDA  2000        J4P  3002
(3002)  MUL  2000        HLT  0
        SLAX 5
```

Объем — 7; время — $171u$.

Действительно минимальное время выполнения. Как указал Р. В Флойд (R. W. Floyd), из условий задачи следует, что $|X| \leq 5$, поэтому минимальное время выполнения достигается при обращении к таблице.

```
(3000)  LD1  2000
        LDA  3500,1
        HLT  0
(3495)  (-5)^13
(3496)  (-4)^13
  .
  .
  .
(3505)  (+5)^13
```

Объем — 14, время — $4u$.

23. Следующее решение, предложенное Р. Д. Диксоном (R. D. Dixon), явно удовлетворяет всем условиям.

```
(3000) ENT1 4                    DEC1 1
(3001) LDA  200                  J1NN 3001
       SRA  0,1                  SLAX 5
       SRAX 1                    HLT  0
```

24. (a) `DIV 3500`, где 3500 = | + | 1 | 0 | 0 | 0 | 0 |.

(b) `SRC 4; SRA 1; SLC 5`.

25. Вот некоторые идеи. (a) Установить более быструю память, больше устройств ввода-вывода. (b) Использовать поле I для индексирования регистра J и/или для кратного индексирования (для определения двух различных индексных регистров), и/или "косвенной адресации" (упр. 2.2.2–3, 4, 5). (c) Расширить индексные регистры и регистр J до полных пяти байтов. Тогда на ячейки со старшими адресами можно будет ссылаться только путем индексирования, но это вполне терпимо при наличии кратного индексирования (см. (b)). (d) Добавить функцию прерывания, воспользовавшись отрицательными адресами памяти, как в упр. 1.4.4–18. (e) Установить "системные часы", снова воспользовавшись отрицательными адресами памяти. (f) К двоичной версии MIX добавить логические операции, переходы к четным и нечетным регистрам и двоичные сдвиги (например, см. упр. 2.5–28, 5.2.2–12 и 6.3–9, а также программы 4.5.2B, 6.4–(24), раздел 7.1). (g) Использовать команду "выполнить" (для команды, находящейся в ячейке M) в качестве еще одного варианта C = 5. (h) Использовать еще один вариант C = 48, ..., 55 — присвоить CI ← регистр : M.

26. Очень заманчиво использовать поле (2:5) для ввода с перфокарты колонок 7–10, но это невозможно, так как $2 \cdot 8 + 5 = 21$. Чтобы читателю легче было следить за выполнением программы, она представлена здесь на символическом языке (в предвосхищении раздела 1.3.2).

					Символы, перфорированные на карте
	BUFF	EQU	29	Буферная область — 0029–0044.	
		ORIG	0		
00	LOC	IN	16(16)	Прочитать вторую перфокарту.	␣0␣06
01	READ	IN	BUFF(16)	Прочитать следующую перфокарту.	␣Z␣06
02		LD1	0(0:0)	rI1 ← 0.	␣␣␣␣I
03		JBUS	*(16)	Ожидать окончания чтения.	␣C␣04
04		LDA	BUFF+1	rA ← колонки 6–10.	␣0␣EH
05	=1=	SLA	1		␣A␣␣F
06		SRAX	6	rAX ← колонки 7–10.	␣F␣CF
07	=30=	NUM	30		␣0␣␣E
08		STA	LOC	LOC ← начальный адрес.	␣␣␣EU
09		LDA	BUFF+1(1:1)		␣0␣IH
10		SUB	=30=(0:2)		␣G␣BB
11	LOOP	LD3	LOC	rI3 ← LOC.	␣␣␣EJ
12		JAZ	0,3	Перейти, если идет карта перехода.	␣␣CA.
13		STA	BUFF	BUFF ← счетчик.	␣Z␣EU
14		LDA	LOC		␣␣␣EH
15		ADD	=1=(0:2)		␣E␣BA
16		STA	LOC	LOC ← LOC + 1.	␣␣␣EU
17		LDA	BUFF+3,1(5:5)		␣2A-H
18		SUB	=25=(0:2)		␣S␣BB
19		STA	0,3(0:0)	Сохранить знак.	␣␣C␣U
20		LDA	BUFF+2,1		␣1AEH
21		LDX	BUFF+3,1		␣2AEN
22	=25=	NUM	25		␣V␣␣E
23		STA	0,3(1:5)	Сохранить абсолютное значение.	␣␣CLU
24		MOVE	0,1(2)	rI1 ← rI1 + 2. (!)	␣␣ABG

```
25      LDA     BUFF                                    ␣Z␣EH
26      SUB     =1=(0:2)    Уменьшить значение счетчика.  ␣E␣BB
27      JAP     LOOP        Повторять до обнуления счетчика. ␣J␣B.
28      JMP     READ        Теперь прочитать новую перфокарту. ␣A␣␣9
```

РАЗДЕЛ 1.3.2

1. ENTX 1000; STX X.

2. Команда STJ из строки 03 переустанавливает этот адрес. (Адрес такой команды принято обозначать через "*", во-первых, для простоты, а во-вторых, потому что это обеспечивает наглядное тестирование программы на случай, если по недосмотру вход в подпрограмму осуществлен некорректно. Заметим, что некоторые предпочитают обозначение "*-*".)

3. Выполняется считывание 100 слов с накопителя на магнитной ленте под номером нуль; максимальное и последнее число меняются местами; максимальное число из оставшихся 99 и последнее из них меняются местами; и т. д. В конце концов все 100 слов будут полностью рассортированы в порядке неубывания. Затем результат будет записан на магнитную ленту (устройство номер один) (ср. с алгоритмом 5.2 3S).

4. Содержимое ненулевых ячеек таково.

3000:	+	0000	00	18	35
3001:	+	2051	00	05	09
3002:	+	2050	00	05	10
3003:	+	0001	00	00	49
3004:	+	0499	01	05	26
3005:	+	3016	00	01	41
3006:	+	0002	00	00	50
3007:	+	0002	00	02	51
3008:	+	0000	00	02	48
3009:	+	0000	02	02	55
3010:	-	0001	03	05	04
3011:	+	3006	00	01	47
3012:	-	0001	03	05	56
3013:	+	0001	00	00	51
3014:	+	3008	00	06	39
3015:	+	3003	00	00	39
3016:	+	1995	00	18	37
3017:	+	2035	00	02	52
3018:	-	0050	00	02	53
3019:	+	0501	00	00	53
3020:	-	0001	05	05	08

3021:	+	0000	00	01	05
3022:	+	0000	04	12	31
3023:	+	0001	00	01	52
3024:	+	0050	00	01	53
3025:	+	3020	00	02	45
3026:	+	0000	04	18	37
3027:	+	0024	04	05	12
3028:	+	3019	00	00	45
3029:	+	0000	00	02	05
0000:	+				2
1995:	+	06	09	19	22	23
1996:	+	00	06	09	25	05
1997:	+	00	08	24	15	04
1998:	+	19	05	04	00	17
1999:	+	19	09	14	05	22
2024:	+					2035
2049:	+					2010
2050:	+					3
2051:	-					499

(Два последних слова можно поменять местами, внеся соответствующие изменения в ячейки 3001 и 3002)

5. Каждая команда OUT ждет окончания предыдущей операции, выполняемой АЦПУ (с другого буфера).

6. (a) Если n — не простое число, то по определению n имеет делитель d, такой, что $1 < d < n$. Если $d > \sqrt[3]{n}$, то n/d — делитель, такой, что $1 < n/d < \sqrt{n}$. (b) Если N не является простым числом, то N имеет *простой* делитель d, такой, что $1 < d \le \sqrt{N}$. Алгоритм подтвердил, что N не имеет простых делителей $\le p =$ PRIME[K]; кроме того, $N = pQ + R < pQ + p \le p^2 + p < (p+1)^2$. Любой простой делитель N, следовательно, больше $p + 1 > \sqrt{N}$.

Необходимо также доказать, что если N простое, то существует достаточно большое простое число, меньшее N, т. е. что $(k+1)$-е простое число p_{k+1} меньше $p_k^2 + p_k$. В противном случае K превышало бы J и PRIME[K] было бы нулем, когда нам понадобилось бы, чтобы оно было большим. Доказательство следует из "постулата Бертрана": если p — простое, то существует простое число, большее p, но меньшее $2p$.

7. (a) Это ссылка на метку строки 29. (b) Программа выполнена не будет; строка 14 будет ссылаться на строку 15, а не на строку 25; строка 24 будет ссылаться на строку 15, а не на строку 12.

8. Печатает 100 строк. Если все 12 000 символов этих строк расположить так, чтобы они примыкали друг к другу, то они займут довольно много места и это будет выглядеть так: пять пробелов, пять букв A, десять пробелов и пять букв A, пятнадцать пробелов, ... $5k$ *пробелов и пять букв* A, $5(k+1)$ *пробелов ... и т. д., пока не будут напечатаны* все 12 000 символов. Третья от конца строка заканчивается последовательностью букв AAAAA и 35 пробелами; последние две строки полностью пусты. Общий результат представляет собой пример манипуляций полем ОП.

9. В поле (4:4) каждого элемента следующей таблицы содержится максимальное значение F, а в поле (1:2) — адрес соответствующей программы проверки допустимости.

```
 B     EQU  1(4:4)          BEGIN  LDA  INST
BMAX   EQU  B-1                   CMPA VALID(3:3)
UMAX   EQU  20                    JG   BAD              Поле I > 6?
TABLE  NOP  GOOD(BMAX)            LD1  INST(5:5)
       ADD  FLOAT(5:5)            DEC1 64
       SUB  FLOAT(5:5)            J1NN BAD              Поле C ≥ 64?
       MUL  FLOAT(5:5)            CMPA TABLE+64,1(4:4)
       DIV  FLOAT(5:5)            JG   BAD              Поле F > максимального F?
       HLT  GOOD                  LD1  TABLE+64,1(1:2)  Перейти к специальной
       SRC  GOOD                  JMP  0,1                программе.
       MOVE MEMORY(BMAX)  FLOAT   CMPA VALID(4:4)       F = 6 допустимо в
       LDA  FIELD(5:5)            JE   GOOD               арифметических
      ...                 FIELD   ENTA 0                  операциях.
       STZ  FIELD(5:5)            LDX  INST(4:4)        Это хитроумный
       JBUS MEMORY(UMAX)          DIV  =9=                способ проверки
       IOC  GOOD(UMAX)            STX  *+1(0:2)           допустимости
       IN   MEMORY(UMAX)          INCA 0                  частичного поля.
       OUT  MEMORY(UMAX)          CMPA =5=
       JRED MEMORY(UMAX)          JG   BAD
       JLE  MEMORY        MEMORY  LDX  INST(3:3)
       JANP MEMORY                JXNZ GOOD             Если I = 0,
      ...                         LDX  INST(0:2)          то адрес
       JXNP MEMORY                JXN  BAD                точно указывает
       ENNA GOOD                  CMPX =3999=             на допустимую
      ...                         JLE  GOOD               ячейку памяти.
```

```
ENNXGOOD                    JMP  BAD
CMPAFLOAT(5:5)         VALID CMPX 3999,6(6)      ▌
CMP1FIELD(5:5)
...
CMPXFIELD(5:5)
```

10. Главная трудность этой задачи состоит в том, что в строке или столбце может быть несколько минимумов или максимумов и каждый из них — это потенциальная седловая точка.

Решение 1. В этом варианте решения по очереди просматриваются все строки, создается список всех столбцов для минимальных элементов строк, а затем проверяется, является ли минимальный элемент строки максимальным элементом столбца. rX ≡ текущий минимум; по мере просмотра матрицы rI1 пробегает значения от 72 до 0, если только не будет найдена седловая точка; rI2 ≡ индекс столбца для элемента из rI1; rI3 ≡ размер списка минимумов. Обратите внимание, что в любом случае цикл заканчивается тогда, когда содержимое индексного регистра ≤ 0.

```
* РЕШЕНИЕ 1
A10     EQU  1008      Адрес a10.
LIST    EQU  1000
START   ENT1 9*8       Начать с правого нижнего угла.
ROWMIN  ENT2 8         Сейчас rI1 "просматривает" восьмой столбец строки.
2H      LDX  A10,1     Кандидат в минимальные элементы строки.
        ENT3 0         Список пуст.
4H      INC3 1
        ST2  LIST,3    Занести индекс столбца в список
1H      DEC1 1         Сместиться на один элемент влево.
        DEC2 1
        J2Z  COLMAX    Просмотр строки закончен?
3H      CMPX A10,1
        JL   1B        Содержимое rX все еще является минимумом?
        JG   2B        Новый минимум?
        JMP  4B        Запомнить новый минимум.
COLMAX  LD2  LIST,3    Взять столбец из списка.
        INC2 9*8-8
1H      CMPX A10,2
        JL   NO        Минимум строки < элемента столбца?
        DEC2 8
        J2P  1B        Просмотр столбца закончен?
YES     INC1 A10+8,2   Да, rI1 ← адрес седловой точки.
        HLT
NO      DEC3 1         Список пуст?
        J3P  COLMAX    Нет, сделать следующую попытку.
        J1P  ROWMIN    Все ли строки просмотрены?
        HLT            Да, rI1 = 0, седловой точки нет.  ▌
```

Решение 2. Добавив немного математики, получаем еще один алгоритм.

Теорема. *Пусть* $R(i) = \min_j a_{ij}$, $C(j) = \max_i a_{ij}$. *Элемент* $a_{i_0j_0}$ *является седловой точкой тогда и только тогда, когда* $R(i_0) = \max_i R(i) = C(j_0) = \min_j C(j)$.

Доказательство. Если $a_{i_0j_0}$ — седловая точка, то для любого фиксированного i выполняется $R(i_0) = C(j_0) \geq a_{i_0j_0} \geq R(i)$, поэтому $R(i_0) = \max_i R(i)$. Аналогично $C(j_0) = \min_j C(j)$.

Обратно, имеем $R(i) \le a_{ij} \le C(j)$ для всех i и j, отсюда $R(i_0) = C(j_0)$ и, следовательно, $a_{i_0 j_0}$ — седловая точка ▮

(Из этого доказательства видно, что всегда выполняется неравенство $\max_i R(i) \le \min_j C(j)$ Поэтому седловой точки не существует тогда и только тогда, когда $R(i)$ строго меньше всех $C(j)$)

Согласно теореме достаточно найти наименьший максимум столбца, а затем — равный ему минимум строки Во время фазы 1 rI1 ≡ индекс столбца, rI2 пробегает по матрице Во время фазы 2 rI1 ≡ возможный ответ, rI2 пробегает по матрице, rI3 ≡ индекс строки, умноженный на 8, rI4 ≡ индекс столбца

```
* РЕШЕНИЕ 2
CMAX    EQU  1000
A10     EQU  CMAX+8
PHASE1  ENT1 8          Начать со столбца 8
3H      ENT2 9*8-8,1
        JMP  2F
1H      CMPX A10,2      В rX по-прежнему максимум?
        JGE  *+2
2H      LDX  A10,2      Новый максимум в столбце
        DEC2 8
        J2P  1B
        STX  CMAX+8,2   Сохранить максимум столбца
        J2Z  1F         Первый раз?
        CMPA CMAX+8,2   В rA еще min max?
        JLE  *+2
1H      LDA  CMAX+8,2
        DEC1 1          Перейти на один столбец влево
        J1P  3B
PHASE2  ENT3 9*8-8      К этому моменту rA = min_j C(j)
3H      ENT2 8,3        Подготовиться к поиску строки
        ENT4 8
1H      CMPA A10,2      min_j C(j) > a[i,j]?
        JG   NO         В этой строке нет седловой точки
        JL   2F
        CMPA CMAX,4     a[i,j] = C(j)?
        JNE  2F
        ENT1 A10,2      Запомнить возможную седловую точку
2H      DEC4 1          Переместиться по строке влево
        DEC2 1
        J4P  1B
        HLT             Седловая точка найдена
NO      DEC3 8
        J3P  3B         Проверить следующую строку
        ENT1 0
        HLT             rI1 = 0, седловой точки нет ▮
```

Предоставляем читателю возможность найти более удачное решение, в котором на этапе 1 записываются все возможные строки, являющиеся кандидатами на проверку наличия седловой точки на этапе 2 Совсем необязательно выполнять поиск во всех строках, достаточно проверить только те строки i_0, для которых $C(j_0) = \min_j C(j)$ влечет $a_{i_0 j_0} = C(j_0)$ Как правило, существует максимум одна такая строка

В некоторых пробных прогонах, когда элементы матрицы выбирались наугад из множества $\{0,1,2,3,4\}$, для нахождения решения методом 1 требовалось время, примерно равное $730u$, а решение методом 2 занимало приблизительно $530u$ времени. Если матрица состоит из одних нулей, то метод 1 позволит найти седловую точку за $137u$, а метод 2 — за $524u$.

Если матрица размера $m \times n$ состоит из *различных* элементов, то задачу можно решить, просмотрев только $O(m+n)$ элементов и выполнив $O(m \log n)$ вспомогательных операций. (См. работу Bienstock, Chung, Fredman, Schäffer, Shor, Suri, *AMM* **98** (1991), 418–419)

11. Предположим, что матрица имеет размер $m \times n$. (a) По теореме, сформулированной в ответе к упр. 10, все седловые точки матрицы имеют одно значение, поэтому (вследствие предположения о том, что элементы матрицы различны) существует максимум одна седловая точка. Согласно свойству симметрии искомая вероятность равна mn, умноженному на вероятность того, что a_{11} является седловой точкой. А эта последняя вероятность равна $1/(mn)!$, умноженному на число перестановок с $a_{12} > a_{11}$, $\ldots$, $a_{1n} > a_{11}$, $a_{11} > a_{21}$, $\ldots$, $a_{11} > a_{m1}$. Это равно $1/(m+n-1)!$, умноженному на число перестановок $m+n-1$ элементов, в которых первый элемент больше следующих $(m-1)$ элементов и меньше оставшихся $(n-1)$ элементов, т. е. равно $(m-1)!\,(n-1)!$. Поэтому в ответе получим

$$mn(m-1)!\,(n-1)!/(m+n-1)! = (m+n)\Big/\binom{m+n}{n}.$$

В нашем случае получаем $17/\binom{17}{8}$, т. е. только один шанс из 1 430. (b) Вследствие второго предположения необходимо использовать совершенно другой метод, так как может быть несколько седловых точек; фактически вся строка (либо весь столбец) должна полностью состоять из седловых точек. Искомая вероятность равна вероятности того, что существует седловая точка с нулевым значением, плюс вероятность того, что существует седловая точка со значением 1. Первое представляет собой вероятность того, что существует по меньшей мере один столбец, полностью состоящий из нулей, а последнее — вероятность того, что существует по меньшей мере одна строка, полностью состоящая из единиц. Поэтому в ответе получаем $\big(1-(1-2^{-m})^n\big)+\big(1-(1-2^{-n})^m\big)$. В нашем случае это будет 92474479623403623 1/18446744073709551616, т. е. приблизительно 1 шанс из из 19.9. Приближенный вид искомой формулы: $n2^{-m}+m2^{-n}$.

12. М. Хофри (M. Hofri) и Ф. Жаке (P. Jacquet) [*Algorithmica* **22** (1998), 516–528] проанализировали случай, когда элементы матрицы размера $m \times n$ различны и выбраны в случайном порядке. Тогда время выполнения двух приведенных программ для MIX составляет соответственно $(6mn+5mH_n+8m+6+5(m+1)/(n-1))u+O((m+n)^2/\binom{m+n}{m})$ и $(5mn+2nH_m+7m+7n+9H_n)u+O(1/n)+O((\log n)^2/m)$ при $m \to \infty$ и $n \to \infty$ в предположении, что $(\log n)/m \to 0$.

13. * ЗАДАЧА ДЕШИФРОВКИ (СЕКРЕТНО)

```
TAPE  EQU  20          Ввести номер устройства.
TYPE  EQU  19          Вывести номер устройства.
SIZE  EQU  14          Ввести размер блока.
OSIZE EQU  14          Вывести размер блока.
TABLE EQU  1000        Таблица результатов
      ORIG TABLE         (первоначально с нулевыми значениями,
      CON  -1            кроме результатов
      ORIG TABLE+46      для пробелов и
      CON  -1            звездочки).
      ORIG 2000
BUF1  ORIG *+SIZE      Первая буферная область.
```

```
        CON   -1          Признак конца буфера.
        CON   *+1         Ссылка на второй буфер.
BUF2    ORIG  *+SIZE      Второй буфер.
        CON   -1          Признак конца буфера.
        CON   BUF1        Ссылка на первый буфер.
BEGIN   IN    BUF1(TAPE)  Ввести первый блок.
        ENT6  BUF2
1H      IN    0,6(TAPE)   Ввести следующий блок.
        LD6   SIZE+1,6    Во время ввода этого блока подготовиться
        ENT5  0,6           к обработке предыдущего.
        JMP   4F
2H      INCA  1                                    } Основной
        STA   TABLE,1     Обновить элементы таблицы. } цикл,
3H      SLAX  1                                    } должен
        STA   *+1(2:2)    rI1 ← следующий символ   } выполняться
        ENT1  0                                    } настолько быстро,
        LDA   TABLE,1                              } насколько это возможно.
        JANN  2B          Нормальный символ?       }
        J1NZ  3F          Звездочка?               }
        JXP   3B          Пропустить пробел.       }
        INC5  1
4H      LDX   0,5         rX ← пять символов.
        JXNN  3B          Перейти, если это не признак конца буфера.
        JMP   1B          Обработка блока закончена.
3H      ENT1  1           Приступить к завершающей фазе. rI1 ← "A".
2H      LDA   TABLE,1
        JANP  1F          Опустить нулевые результаты.
        CHAR              Преобразовать в десятичную форму.
        JBUS  *(TYPE)     Ждать готовности терминала.
        ST1   CHAR(1:1)
        STA   CHAR(4:5)
        STX   FREQ
        OUT   ANS(TYPE)   Вывести один ответ.
1H      CMP1  =63=
        INC1  1           Подсчитано до 63 кодов
        JL    2B            символов.
        HLT
ANS     ALF               Буфер вывода.
        ALF
CHAR    ALF   C  NN
FREQ    ALF   NNNNN
        ORIG  ANS+OSIZE   Оставшаяся часть буфера пуста
        END   BEGIN       Здесь получаем литеральную константу =63=.  ▌
```

В этой задаче буферизация *вывода* нежелательна, так как она позволяет сэкономить максимум $7u$ времени на строку выходной информации.

14. Чтобы сделать эту задачу более сложной и интересной, в приведенном ниже решении, частично принадлежащем Дж. Петолино (J. Petolino), используется множество различных *ухищрений*, позволяющих сократить время выполнения. Может ли читатель сэкономить еще несколько микросекунд?

```
* ДАТА ПАСХИ
EASTER   STJ  EASTX
         STX  Y
         ENTA 0                E1.
         DIV  =19=
         STX  GMINUS1(0:2)
         LDA  Y                E2.
         MUL  =1//100+1=       (См. ниже.)
         INCA 61
         STA  CPLUS60(1:2)
         MUL  =3//4+1=
         STA  XPLUS57(1:2)
CPLUS60  ENTA *
         MUL  =8//25+1=        rA ← Z + 24.
GMINUS1  ENT2 *                E5.
         ENT1 1,2              rI1 ← G.
         INC2 1,1
         INC2 0,2
         INC2 0,1
         INC2 0,2
         INC2 773,1            rI2 ← 11G + 773.
XPLUS57  INCA -*,2             rA ← 11G + Z − X + 20 + 24 · 30 (≥ 0).
         SRAX 5
         DIV  =30=             rX ← E.
         DECX 24
         JXN  4F
         DECX 1
         JXP  2F
         JXN  3F
         DEC1 11
         J1NP 2F
3H       INCX 1
2H       DECX 29               E6.
4H       STX  20MINUSN(0:2)
         LDA  Y                E4.
         MUL  =1//4+1=
         ADD  Y
         SUB  XPLUS57(1:2)     rA ← D − 47.
20MINUSN ENN1 *
         INCA 67,1             E7.
         SRAX 5                rX ← D + N.
         DIV  =7=
         SLAX 5
         DECA -4,1             rA ← 31 − N.
         JAN  1F               E8.
         DECA 31
         CHAR
         LDA  MARCH
         JMP  2F
1H       CHAR
```

```
          LDA   APRIL
2H        JBUS  *(18)
          STA   MONTH
          STX   DAY(1:2)
          LDA   Y
          CHAR
          STX   YEAR
          OUT   ANS(18)          П
EASTX     JMP   *
MARCH     ALF   MARCH
APRIL     ALF   APRIL
ANS       ALF
DAY       ALF   DD
MONTH     ALF   MMMMM
          ALF   ,
YEAR      ALF   YYYYY
          ORIG  *+20
BEGIN     ENTX  1950             "Управляющая"
          ENT6  1950-2000          программа,
          JMP   EASTER             использующая
          INC6  1                  приведенную выше
          ENTX  2000,6             подпрограмму.
          J6NP  EASTER+1
          HLT
          END   BEGIN            ▮
```

Для строгого обоснования перехода в нескольких местах от деления к умножению можно использовать тот факт, что число в rA не слишком велико. Данная программа работает при любых размерах байта.

[Чтобы вычислить дату пасхи для годов, номера которых ≤ 1582, обратитесь к *CACM* **5** (1962), 209–210. Первым систематическим алгоритмом для вычисления даты пасхи были *канонические пасхалии* Викториуса Аквитанского (Victorius of Aquitania) (457 г. н. э.). Существует много подтверждений тому, что единственным нетривиальным применением арифметики в Европе на протяжении средних веков было вычисление даты пасхи, поэтому подобные алгоритмы имеют историческое значение. Более подробно об этом говорится в книге T. H. O'Beirne, *Puzzles and Paradoxes* (London: Oxford University Press, 1965), Chapter 10; в книге N. Dershowitz, E. M. Reingold, *Calendrical Calculations* (Cambridge Univ. Press, 1997), описываются всевозможные алгоритмы, имеющие отношение к датам.]

15. Первым таким годом является 10 317 г. н. э., хотя описанная ошибка *почти* приводит к проблемам для годов н. э. $10108 + 19k$, где $0 \le k \le 10$.

Между прочим, Т. Х. О'Берн (T. H O'Beirne) указал, что даты пасхи повторяются с периодом 5 700 000 лет. Расчеты, проведенные Робертом Хиллом (Robert Hill), показали, что самой частой датой пасхи является 19 апреля (220 400 раз в течение указанного периода), самой ранней и наименее частой — 22 марта (27 550 раз), а самой поздней и предпоследней по частоте — 25 апреля (42 000 раз). Хилл нашел изящное объяснение того интересного факта, что частота каждой конкретной даты пасхи в указанном периоде времени всегда кратна 25.

16. Будем работать с масштабированными числами, $R_n = 10^n r_n$. В этом случае $R_n(1/m) = R$ тогда и только тогда, когда $10^n/(R+\frac{1}{2}) < m \le 10^n/(R-\frac{1}{2})$; отсюда находим $m_h = \lfloor 2 \cdot 10^n/(2R-1) \rfloor$.

```
* СУММА ГАРМОНИЧЕСКОГО РЯДА
BUF   ORIG *+24
START ENT2 0
      ENT1 3          5 - n.
      ENTA 20
OUTER MUL  =10=
      STX  CONST      2 · 10^n.
      DIV  =2=
      ENTX 2
      JMP  1F
INNER STA  R
      ADD  R
      DECA 1
      STA  TEMP       2R - 1.
      LDX  CONST
      ENTA 0
      DIV  TEMP
      INCA 1
      STA  TEMP       m_h + 1.
      SUB  M
      MUL  R
      SLAX 5
      ADD  S
      LDX  TEMP
1H    STA  S          Частичная сумма.
      STX  M          m = m_e.
      LDA  M
      ADD  M
      STA  TEMP
      LDA  CONST
      ADD  M          Вычислить R = R_n(1/m) =
      SRAX 5            ⌊(2 · 10^n + m)/(2m)⌋.
      DIV  TEMP
      JAP  INNER      R > 0?
      LDA  S          10^n S_n.
      CHAR
      SLAX 0,1        Четкое форматирование.
      SLA  1
      INCA 40         Десятичная точка.
      STA  BUF,2
      STX  BUF+1,2
      INC2 3
      DEC1 1
      LDA  CONST
      J1NN OUTER
      OUT  BUF(18)
      HLT
      END  START
```

Выходные данные

```
0006.16        0008.449        0010.7509        0013.05363
```

получены за $65595u$ плюс время вывода. (Было бы быстрее вычислить $R_n(1/m)$ непосредственно при $m < 10^{n/2}\sqrt{2}$, а затем применить предлагаемую процедуру.)

17. Положим $N = \lfloor 2\cdot 10^n/(2m\pm 1)\rfloor$. Тогда $S_n = H_N + O(N/10^n) + \sum_{k=1}^{m}(\lfloor 2\ 10^n/(2k-1)\rfloor - \lfloor 2\cdot 10^n/(2k+1)\rfloor)k/10^n = H_N + O(m^{-1}) + O(m/10^n) - 1 + 2H_{2m} - H_m = n\ln 10 + 2\gamma - 1 + 2\ln 2 + O(10^{-n/2})$, если просуммировать по частям и положить $m \approx 10^{n/2}$.

Между прочим, следующие несколько значений — это $S_6 = 15.356262$, $S_7 = 17.6588276$, $S_8 = 19.96140690$, $S_9 = 22.263991779$ и $S_{10} = 24.5665766353$; приближенным значением для S_{10} будет ≈ 24.566576621, которое точнее расчетного.

18.
```
FAREY STJ  9F      Предполагаем, что в rI1 содержится n, где n > 1.
      STZ  X       x_0 ← 0.
      ENTX 1
      STX  Y       y_0 ← 1.
      STX  X+1     x_1 ← 1.
      ST1  Y+1     y_1 ← n.
      ENT2 0       k ← 0.
1H    LDX  Y,2
      INCX 0,1
      ENTA 0
      DIV  Y+1,2
      STA  TEMP    ⌊(y_k + n)/y_{k+1}⌋.
      MUL  Y+1,2
      SLAX 5
      SUB  Y,2
      STA  Y+2,2   y_{k+2}.
      LDA  TEMP
      MUL  X+1,2
      SLAX 5
      SUB  X,2
      STA  X+2,2   x_{k+2}.
      CMPA Y+2,2   Проверить условие x_{k+2} < y_{k+2}.
      INC2 1       k ← k + 1.
      JL   1B      Если да, продолжить.
9H    JMP  *       Выйти из подпрограммы. ∎
```

19. (a) По индукции. (b) Пусть $k \geq 0$ и $X = ax_{k+1} - x_k$, $Y = ay_{k+1} - y_k$, где $a = \lfloor (y_k + n)/y_{k+1}\rfloor$. Из (a) и того, что $0 < Y \leq n$, получаем $X \perp Y$ и $X/Y > x_{k+1}/y_{k+1}$. Поэтому, если $X/Y \neq x_{k+2}/y_{k+2}$, то по определению $X/Y > x_{k+2}/y_{k+2}$. Отсюда следует, что

$$\begin{aligned}\frac{1}{Yy_{k+1}} &= \frac{Xy_{k+1} - Yx_{k+1}}{Yy_{k+1}} = \frac{X}{Y} - \frac{x_{k+1}}{y_{k+1}} \\ &= \left(\frac{X}{Y} - \frac{x_{k+2}}{y_{k+2}}\right) + \left(\frac{x_{k+2}}{y_{k+2}} - \frac{x_{k+1}}{y_{k+1}}\right) \\ &\geq \frac{1}{Yy_{k+2}} + \frac{1}{y_{k+1}y_{k+2}} = \frac{y_{k+1} + Y}{Yy_{k+1}y_{k+2}} > \frac{n}{Yy_{k+1}y_{k+2}} \geq \frac{1}{Yy_{k+1}}.\end{aligned}$$

Исторические замечания. К Гарос (C Haros) предложил более сложное правило построения таких последовательностей в работе *J. de l'École Polytechnique* **4**, 11 (1802), 364–368; его метод правилен, но доказательство несовершенно. Несколькими годами позже геолог Джон Фарей (John Farey) независимо предположил, что x_k/y_k всегда равно $(x_{k-1} + x_{k+1})/(y_{k-1} + y_{k+1})$ [*Philos. Magazine and Journal* **47** (1816), 385–386], доказательство вскоре

было предоставлено О Коши (A. Cauchy) [*Bull. Société Philomathique de Paris* (3) **3** (1816), 133–135], который дал этому ряду имя Фарея О других интересных свойствах данных рядов говорится в работе G. H Hardy, E. M Wright, *An Introduction to the Theory of Numbers*, Chapter 3.

20. * ЗАДАЧА О СИГНАЛАХ СВЕТОФОРА

```
BSIZE   EQU  1(4:4)          Размер байта.
2BSIZE  EQU  2(4:4)          Двойной размер байта.
DELAY   STJ  1F              Если в rA содержится n,
        DECA 6                 эта подпрограмма
        DECA 2                 ожидает ровно max(n,7)u
        JAP  *-1               времени, не считая
        JAN  *+2               времени на переход к подпрограмме
        NOP
1H      JMP  *
FLASH   STJ  2F          4   Это подпрограмма мигания
        ENT2 8           5     соответствующего сигнала СТОЙТЕ.
1H      LDA  =49991=     7
        JMP  DELAY       8
        DECX 0,1         9   Выключение сигнала.
        LDA  =49996=     2
        JMP  DELAY       3
        INCX 0,1         4   СТОЙТЕ.
        DEC2 1           1
        J2Z  1F          2   Повторить восемь раз.
        LDA  =49993=     4
        JMP  1B          5   Вернуться к началу цикла.
1H      LDA  =399992=    4   Установить желтый через 2u после выхода.
        JMP  DELAY       5
2H      JMP  *           6
WAIT    JNOV *           5   Зеленый свет на Дель-Мар до момента переключения.
TRIP    INCX BSIZE       6   На Дель-Мар сигнал СТОЙТЕ.
        ENT1 2BSIZE      1
        JMP  FLASH       2   Мигание сигнала на Дель-Мар.
        LDX  BAMBER      8   Желтый свет на проспекте.
        LDA  =799995=    2
        JMP  DELAY       3   Ожидать 8 с.
        LDX  AGREEN      5   Зеленый свет на авеню.
        LDA  =799996=    2
        JMP  DELAY       3   Ожидать 8 с
        INCX 1           4   На Беркли сигнал СТОЙТЕ.
        ENT1 2           1
        JMP  FLASH       2   Мигание сигнала на Беркли.
        LDX  AAMBER      8   Желтый свет на авеню
        JOV  *+1         1   Отменить лишнее переключение.
        LDA  =499994=    3
        JMP  DELAY       4   Ожидать 5 с.
BEGIN   LDX  BGREEN      6   Зеленый свет на проспекте.
        LDA  =1799994=   2
        JMP  DELAY       3   Ожидать минимум
        JMP  WAIT        4     18 с.
```

```
AGREEN ALF   CABA        Зеленый свет для авеню.
AAMBER ALF   CBBB        Желтый свет для авеню.
BGREEN ALF   ACAB        Зеленый свет на проспекте.
BAMBER ALF   BCBB        Желтый свет на проспекте.
       END   BEGIN
```

22.
```
* ЗАДАЧА ИОСИФА
N   EQU  24
M   EQU  11
X   ORIG *+N
OH  ENT1 N-1          1          Установить в каждой ячейке
    STZ  X+N-1        1            номер следующего человека
    ST1  X-1,1        N-1          в последовательности.
    DEC1 1            N-1
    J1P  *-2          N-1
    ENTA 1            1          (Сейчас rI1 = 0.)
1H  ENT2 M-2          N-1        (Предполагаем, что M > 2.)
    LD1  X,1          (M-2)(N-1) Отсчет
    DEC2 1            (M-2)(N-1)   по кругу.
    J2P  *-2          (M-2)(N-1)
    LD2  X,1          N-1        rI1 ≡ счастливчик.
    LD3  X,2          N-1        rI2 ≡ обреченный.
    CHAR              N-1        rI3 ≡ следующий.
    STX  X,2(4:5)     N-1        Сохранить номер казненного.
    NUM               N-1
    INCA 1            N-1
    ST3  X,1          N-1        Взять человека из круга.
    ENT1 0,3          N-1
    CMPA =N=          N-1
    JL   1B           N-1
    CHAR              1          Остался один человек;
    STX  X,1(4:5)     1            его тоже казнят.
    OUT  X(18)        1          Напечатать ответ.
    HLT               1
    END  0B
```

Последним остается человек в позиции 15. Общее время выполнения программы без печати результатов составляет $(4(N-1)(M+7\,5)+16)u$ Программу можно несколько улучшить, например воспользоваться предложением Д. Инголза (D. Ingalls) и взять состоящие из трех слов пакеты кода "`DEC2 1; J2P NEXT; JMP OUT`", где команда `OUT` модифицирует поле `NEXT` таким образом, что удаляет пакет Асимптотически более быстрый метод приведен в упр. 5.1.1–5.

РАЗДЕЛ 1.3.3

1. (1 2 4)(3 6 5).

2. $a \leftrightarrow c$, $c \leftrightarrow f$; $b \leftrightarrow d$. Совершенно очевидно, как это обобщить на случай произвольной перестановки.

3. $\begin{pmatrix} a & b & c & d & e & f \\ d & b & f & c & a & e \end{pmatrix}$.

4. $(a\,d\,c\,f\,e)$.

5. 12 (см упр 20).

6. Общее время увеличится на $4u$ для каждого пустого ввода, перед которым идет непустое слово "(", и еще на $5\dot{u}$ для каждого пустого слова, перед которым идет непустое слово-имя Начальные пробелы и пробелы между циклами не оказывают влияния на время выполнения программы Положение пробелов никак не влияет на время выполнения программы B

7. $X = 2$, $Y = 29$, $M = 5$, $N = 7$, $U = 3$, $V = 1$ Общее время выполнения согласно (18) равно $2161u$

8. Да В этом случае нужно использовать обратную перестановку, чтобы x_i переходило в x_j тогда и только тогда, когда $T[j] = i$ (Тогда окончательный вид циклической формы будет построен справа налево с помощью таблицы T)

9. Нет Например, если вводом является (6), то программа A в качестве вывода даст "`(ADG)(CEB)`", а программа B — "`(CEB)(DGA)`" Эти ответы эквивалентны, но не идентичны, так как циклическая запись не единственна В программе A первым элементом цикла выбирается крайний слева, а в программе B — последний отличный от других при движении справа налево

10. (1) По закону Кирхгофа получаем $A = 1 + C - D$, $B = A + J + P - 1$, $C = B - (P - L)$, $E = D - L$, $G = E$, $Q = Z$, $W = S$ (2) Объяснение B = число слов ввода $= 16X - 1$, C = число непустых слов $= Y$, $D = C - M$, $E = D - M$, F = число сравнений при поиске по таблице имен, $H = N$, $K = M$, $Q = N$, $R = U$, $S = R - V$, $T = N - V$, так как каждое из других имен помечено (3) В итоге получаем $(4F + 16Y + 80X + 21N - 19M + 9U - 16V)u$ Это несколько лучший результат, чем дает программа A, поскольку, безусловно, F меньше, чем $16NX$ Общее время выполнения в данном случае равно $983u$, так как $F = 74$

11. "Отразите" ее Например, обратной к перестановке $(a\,c\,f)(b\,d)$ является $(d\,b)(f\,c\,a)$

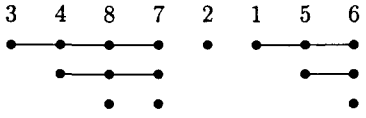
12. (a) Значение, которое находится в ячейке $L + mn - 1$, во время транспонирования остается на месте, поэтому ее можно исключить из рассмотрения A в остальном, если $x = n(i - 1) + (j - 1) < mn - 1$, значение из ячейки $L + x$ должно перейти в ячейку $L + mx \bmod N = L + (mn(i - 1) + m(j - 1)) \bmod N = L + m(j - 1) + (i - 1)$, так как $mn \equiv 1$ (по модулю N) и $0 \le m(j - 1) + (i - 1) < N$ (b) Если в каждой ячейке памяти один бит свободен (например, бит знака), элементы можно "помечать" по мере их перемещения с помощью алгоритма, подобного алгоритму I [См M F Berman, *JACM* **5** (1958), 383–384] Если для битов пометки нет свободного места, то их можно сохранять во вспомогательной таблице или использовать список элементов, содержащихся во всех не единичных циклах Для каждого делителя d числа N можно по отдельности транспонировать те элементы, которые кратны d, так как m взаимно просто с N Длина цикла, содержащего x, где $\gcd(x, N) = d$, — это наименьшее целое $r > 0$, такое, что $m^r \equiv 1$ (по модулю N/d) Для каждого d нужно найти $\varphi(N/d)/r$ элементов-представителей, по одному из каждого цикла Это можно сделать с помощью ряда теоретико-множественных методов, но они недостаточно просты для того, чтобы мы могли ими удовлетвориться Эффективный, но довольно сложный алгоритм можно получить, применяя теорию чисел, а также используя небольшую таблицу битов пометок [См N Brenner, *CACM* **16** (1973), 692–694] И наконец, существует метод, который является аналогом алгоритма J Он работает медленнее, но зато не требует дополнительной памяти и *любые* нужные перестановки выполняет *in situ** [См P F Windley, *Comp J* **2** (1959), 47–48, D E Knuth, *Proc IFIP Congress* (1971), **1**, 19–27, E G Cate, D W Twigg, *ACM Trans Math Software* **3** (1977), 104–110, F E Fich, J I Munro, P V Poblete, *SICOMP* **24** (1995), 266–278]

13. Покажите по индукции, что в начале шага J2 $X[i] = +j$ тогда и только тогда, когда $j > m$ и j переходит в i в результате перестановки π $X[i] = -j$ тогда и только тогда, когда

* "На месте" (лат) — *Прим перев*

i переходит в j в результате перестановки π^{k+1}, где k — наименьшее неотрицательное целое, такое, что π^k переводит i в число, которое $\le m$.

14. Запишем *обратную* перестановку к заданной в канонической циклической форме и уберем скобки. Величина $A-N$ — это сумма последовательных элементов, которые больше заданного и находятся непосредственно справа от него. Например, если исходной является перестановка (1 6 5)(3 7 8 4), то канонической формой обратной перестановки будет (3 4 8 7)(2)(1 5 6). Введем массив



Величина A — это количество "точек", т. е. она равна 16. Число точек под k-м элементом — это *количество минимумов справа налево среди первых k элементов* (в приведенном выше примере под элементом 7 находятся 3 точки, так как в последовательности 3 487 всего 3 минимума справа налево). Значит, среднее равно $H_1 + H_2 + \cdots + H_n = (n+1)H_n - n$.

15. Если первым символом линейного представления является 1, то последним символом канонического представления является 1. Если первым символом линейного представления является некоторое $m > 1$, то в каноническом представлении появляется "$\ldots 1m \ldots$". Поэтому единственным решением является перестановка одного объекта. (Можно считать, что существует также перестановка из *пустого множества* объектов.

16. 1324, 4231, 3214, 4213, 2143, 3412, 2413, 1243, 3421, 1324, ...

17. (a) Вероятность того, что этим циклом является m-цикл, равна $n!/m$, деленному на $n!\,H_n$, т. е. $p_m = 1/(mH_n)$. Средняя длина цикла равна $p_1 + 2p_2 + 3p_3 + \cdots = \sum_{m=1}^{n}(m/mH_n) = n/H_n$ (b) Поскольку общее число m-циклов равно $n!/m$, общее число появлений элементов в m-циклах равно $n!$. Согласно свойству симметрии каждый элемент появляется так же часто, как и любой другой, поэтому k появляется в m-циклах $n!/n$ раз. Следовательно, в *этом* случае $p_m = 1/n$ для всех k и m; среднее равно $\sum_{m=1}^{n} m/n = (n+1)/2$.

18. См. упр. 22, (e).

19. $|P_{n0} - n!/e| = 1/(n+1)! - 1/(n+2)! + \cdots$, т. е. сумме знакопеременного ряда, члены которого убывают по модулю. Эта сумма меньше, чем $1/(n+1)! \le \frac{1}{2}$.

20. Существует всего $\alpha_1 + \alpha_2 + \cdots$ циклов, которые можно переставлять между собой, причем каждый отдельный m-цикл можно независимо от других записать m способами. Поэтому ответом будет

$$(\alpha_1 + \alpha_2 + \cdots)!\, 1^{\alpha_1} 2^{\alpha_2} 3^{\alpha_3} \ldots .$$

21. $1/(\alpha_1!\, 1^{\alpha_1} \alpha_2!\, 2^{\alpha_2} \ldots)$, если $n = \alpha_1 + 2\alpha_2 + \cdots$; в противном случае вероятность равна 0.

Доказательство. Выпишите в ряд α_1 циклов длины 1, α_2 циклов длины 2 и т. д., вместо элементов вставив пробелы. Например, если $\alpha_1 = 1$, $\alpha_2 = 2$, $\alpha_3 = \alpha_4 = \cdots = 0$, то получим "(-) (--) (--)". Заполните пустые позиции всеми $n!$ возможными способами. Тогда каждая перестановка нужного вида появится ровно $\alpha_1!\, 1^{\alpha_1} \alpha_2!\, 2^{\alpha_2} \ldots$ раз.

22. (a) Если $k_1 + 2k_2 + \cdots = n$, то вероятность в п. (ii) равна $\prod_{j\ge 0} f(w, j, k_j)$, что по предположению равно $(1-w)w^n/k_1!\, 1^{k_1} k_2!\, 2^{k_2} \ldots$. Отсюда

$$\frac{f(w,\, m,\, k_m+1)}{f(w, m, k_m)} = \left(\prod_{j\ge 0} f(w, j, k_j)\right)^{-1} \prod_{j\ge 0} f(w,\, j,\, k_j + \delta_{jm}) = \frac{w^m}{m(k_m+1)}.$$

Тогда по индукции получим

$$f(w,m,k) = \frac{1}{k!}\left(\frac{w^m}{m}\right)^k f(w,m,0).$$

Теперь из условия (i) следует, что

$$f(w,m,k) = \frac{1}{k!}\left(\frac{w^m}{m}\right)^k e^{-w^m/m}.$$

[Другими словами, α_m выбирается из распределения Пуассона; см. упр. 1.2.10–15.]

(b) $$\sum_{\substack{k_1+2k_2+\cdots=n\\ k_1,k_2,\ldots\ge 0}} \left(\prod_{j\ge 0} f(w,j,k_j)\right) = (1-w)w^n \sum_{\substack{k_1+2k_2+\cdots=n\\ k_1,k_2,\ldots\ge 0}} P(n;k_1,k_2,\ldots)$$
$$= (1-w)w^n.$$

Следовательно, вероятность того, что $\alpha_1 + 2\alpha_2 + \cdots \le n$, равна $(1-w)(1+w+\cdots+w^n) = 1-w^{n+1}$.

(c) Среднее ϕ равно

$$\sum_{n\ge 0}\left(\sum_{k_1+2k_2+\cdots=n} \phi(k_1,k_2,\ldots)\Pr(\alpha_1=k_1,\alpha_2=k_2,\ldots)\right)$$
$$= (1-w)\sum_{n\ge 0} w^n\left(\sum_{k_1+2k_2+\cdots=n} \phi(k_1,k_2,\ldots)/k_1!\,1^{k_1}k_2!\,2^{k_2}\ldots\right).$$

(d) Пусть $\phi(\alpha_1,\alpha_2,\ldots) = \alpha_2+\alpha_4+\alpha_6+\cdots$. Среднее значение линейной комбинации ϕ равно сумме средних значений $\alpha_2,\alpha_4,\alpha_6,\ldots$. Среднее значение α_m равно

$$\sum_{k\ge 0} kf(w,m,k) = \sum_{k\ge 1}\frac{1}{(k-1)!}\left(\frac{w^m}{m}\right)^k e^{-w^m/m} = \frac{w^m}{m}.$$

Следовательно, среднее значение ϕ равно

$$\frac{w^2}{2}+\frac{w^4}{4}+\frac{w^6}{6}+\cdots = \frac{1-w}{2}(H_1w^2+H_1w^3+H_2w^4+H_2w^5+H_3w^6+\cdots).$$

Искомым ответом будет $\frac{1}{2}H_{\lfloor n/2\rfloor}$.

(e) Положим $\phi(\alpha_1,\alpha_2,\ldots) = z^{\alpha_m}$ и заметим, что среднее значение ϕ равно

$$\sum_{k\ge 0} f(w,m,k)z^k = \sum_{k\ge 0}\frac{1}{k!}\left(\frac{w^m z}{m}\right)^k e^{-w^m/m} = e^{w^m(z-1)/m} = \sum_{j\ge 0}\frac{w^{mj}}{j!}\left(\frac{z-1}{m}\right)^j$$
$$= (1-w)\sum_{n\ge 0} w^n\left(\sum_{0\le j\le n/m}\frac{1}{j!}\left(\frac{z-1}{m}\right)^j\right)$$
$$= (1-w)\sum_{n\ge 0} w^n G_{nm}(z).$$

Отсюда

$$G_{nm}(z) = \sum_{0\le j\le n/m}\frac{1}{j!}\left(\frac{z-1}{m}\right)^j; \quad p_{nkm} = \frac{1}{m^k k!}\sum_{0\le j\le n/m-k}\frac{(-1/m)^j}{j!}.$$

Статистическими характеристиками будут (min 0, ave $1/m$, max $\lfloor n/m\rfloor$, dev $\sqrt{1/m}$), где $n \ge 2m$.

23. Постоянная λ равна $\int_0^\infty \exp(-t - E_1(t))\,dt$, где $E_1(x) = \int_x^\infty e^{-t}dt/t$. [См. *Trans. Amer. Math. Soc.* **121** (1966), 340–357, где доказывается множество других фактов, в частности, что средняя длина *самого короткого* цикла приближенно равна $e^{-\gamma}\ln n$.] Следующие члены асимптотического представления l_n были найдены Ксавье Гордоном (Xavier Gourdon). Первые члены ряда выглядят так:

$$\lambda n + \tfrac{1}{2}\lambda - \tfrac{1}{24}e^{\gamma}n^{-1} + (\tfrac{1}{48}e^{\gamma} - \tfrac{1}{8}(-1)^n)n^{-2} + (\tfrac{17}{3840}e^{\gamma} + \tfrac{1}{8}(-1)^n + \tfrac{1}{6}\omega^{1-n} + \tfrac{1}{6}\omega^{n-1})n^{-3},$$

где $\omega = e^{2\pi i/3}$. Вильям Ч. Митчелл (William C Mitchell) с высокой точностью вычислил значение $\lambda = .62432\ 99885\ 43550\ 87099\ 29363\ 83100\ 83724\ 41796+$ [*Math. Comp.* **22** (1968), 411–415]. Пока неизвестны какие-либо соотношения, связывающие λ с классическими математическими константами. Тем не менее эта же константа была вычислена в другом контексте Карлом Дикманом (Karl Dickman) в работе *Arkiv för Mat., Astron. och Fys.* **22A**, 10 (1930), 1–14. Но совпадение заметили только спустя много лет [*Theor Comp. Sci.* **3** (1976), 373].

24. См. D E Knuth, *Proc. IFIP Congress (1971)*, **1**, 19–27.

25. Одно доказательство (индукцией по N) основано на том, что когда N-й элемент является членом s множеств, он добавляет к сумме в точности следующую величину:

$$\binom{s}{0} - \binom{s}{1} + \binom{s}{2} - \cdots = (1-1)^s = \delta_{s0}.$$

Другое доказательство (индукцией по M) основано на том, что число элементов, принадлежащих S_M, но не принадлежащих $S_1 \cup \cdots \cup S_{M-1}$, равно

$$|S_M| - \sum_{1\le j<M} |S_j \cap S_M| + \sum_{1\le j<k<M} |S_j \cap S_k \cap S_M| - \cdots.$$

26. Пусть $N_0 = N$ и

$$N_k = \sum_{1\le j_1< \ <j_k\le M} |S_{j_1} \cap \cdots \cap S_{j_k}|.$$

Тогда искомой формулой будет

$$N_r - \binom{r+1}{r}N_{r+1} + \binom{r+2}{r}N_{r+2} - \cdots.$$

Это можно вывести из самого принципа включения и исключения либо воспользоваться формулой

$$\binom{r}{r}\binom{s}{r} - \binom{r+1}{r}\binom{s}{r+1} + \cdots = \binom{s}{r}\binom{s-r}{0} - \binom{s}{r}\binom{s-r}{1} + \cdots = \delta_{sr},$$

как в упр. 25.

27. Пусть S_j — это числа из заданного интервала, кратные m_j, и пусть $N = am_1 \ldots m_t$. Тогда $|S_j \cap S_k| = N/m_j m_k$ и т. д. Поэтому ответом будет

$$N - N\sum_{1\le j\le t}\frac{1}{m_j} + N\sum_{1\le j<k\le t}\frac{1}{m_j m_k} - \cdots = N\left(1-\frac{1}{m_1}\right)\cdots\left(1-\frac{1}{m_t}\right).$$

Это также будет решением упр. 1.2.4–30, если принять, что $m_1, \ldots, m_t$ — простые числа, являющиеся делителями N.

29. Пропуская человека, присвойте ему новый номер (начиная с $n+1$). Тогда k-м казненным будет номер $2k$, а человек с номером j для $j > n$ прежде имел номер $(2j) \bmod (2n+1)$.

31. См. *CMath*, раздел 3.3.

32. (a) На самом деле $k-1 \le \pi_k \le k+2$, если k — четное, и $k-2 \le \pi_k \le k+1$, если k — нечетное. (b) Выбирайте экспоненты слева направо, полагая $\epsilon_k = 1$ тогда и только тогда, когда k и $k+1$ пока еще находятся в разных циклах перестановки. [Steven Alpern, *J. Combinatorial Theory* **B25** (1978), 62–73.]

33. Для $l = 0$ положим $(\alpha_{01}, \alpha_{02}; \beta_{01}, \beta_{02}) = (\pi, \rho; \epsilon, \epsilon)$ и $(\alpha_{11}, \alpha_{12}; \beta_{11}, \beta_{12}) = (\epsilon, \epsilon; \pi, \rho)$, где $\pi = (1\,4)(2\,3)$, $\rho = (1\,5)(2\,4)$ и $\epsilon = ()$.

Предположим, мы построили такую конструкцию для некоторого $l \ge 0$, где $\alpha_{jk}^2 = \beta_{jk}^2 = ()$ для $0 \le j < m$ и $1 \le k \le n$. Тогда перестановки

$$\begin{aligned}(A_{(jm+j')1}, \ldots, A_{(jm+j')(4n)}; B_{(jm+j')1}, \ldots, B_{(jm+j')(4n)}) = \\ (\sigma^- \alpha_{j1}\sigma, \ldots, \sigma^- \alpha_{jn}\sigma,\ \tau^- \alpha_{j'1}\tau, \ldots, \tau^- \alpha_{j'n}\tau, \\ \sigma^- \beta_{jn}\sigma, \ldots, \sigma^- \beta_{j1}\sigma,\ \tau^- \beta_{j'n}\tau, \ldots, \tau^- \beta_{j'1}\tau; \\ \sigma^- \beta_{j1}\sigma, \ldots, \sigma^- \beta_{jn}\sigma,\ \tau^- \beta_{j'1}\tau, \ldots, \tau^- \beta_{j'n}\tau, \\ \sigma^- \alpha_{jn}\sigma, \ldots, \sigma^- \alpha_{j1}\sigma,\ \tau^- \alpha_{j'n}\tau, \ldots, \tau^- \alpha_{j'1}\tau)\end{aligned}$$

обладают свойством

$$\begin{aligned}A_{(im+i')1} B_{(jm+j')1} \cdots A_{(im+i')(4n)} B_{(jm+j')(4n)} = \\ \sigma^-(1\,2\,3\,4\,5)\sigma\,\tau^-(1\,2\,3\,4\,5)\tau\,\sigma^-(5\,4\,3\,2\,1)\sigma\,\tau^-(5\,4\,3\,2\,1)\tau,\end{aligned}$$

если $i = j$ и $i' = j'$, в противном случае произведение равно $()$. Выбирая $\sigma = (2\,3)(4\,5)$ и $\tau = (3\,4\,5)$, получим искомое произведение $(1\,2\,3\,4\,5)$ при $im + i' = jm + j'$.

Метод построения от l к $l+1$ предложен Дэвидом А. Бэррингтоном (David A. Barrington) [*J. Comp. Syst. Sci.* **38** (1989), 150–164], который доказал общую теорему о том, что любую булеву функцию можно представить в виде произведения перестановок множества $\{1, 2, 3, 4, 5\}$. С помощью аналогичной конструкции можно, например, найти последовательности перестановок $(\alpha_{j1}, \ldots, \alpha_{jn}; \beta_{j1}, \ldots, \beta_{jn})$, такие, что

$$\alpha_{i1}\beta_{j1}\alpha_{i2}\beta_{j2}\ldots\alpha_{in}\beta_{jn} = \begin{cases}(1\,2\,3\,4\,5), & \text{если } i < j; \\ (), & \text{если } i \ge j;\end{cases}$$

для $0 \le i, j < m = 2^{2^l}$, где $n = 6^{l+1} - 4^{l+1}$.

34. Пусть $N = m + n$. Если $m \perp n$, существует только один цикл, так как каждый элемент можно записать в виде $am \bmod N$ для некоторого целого a. В общем случае, если $d = \gcd(m, n)$, то существует ровно d циклов C_0, C_1, $\ldots$, C_{d-1}, где C_j содержит элементы $\{j, j+d, \ldots, j+N-d\}$, расположенные в некотором порядке. Тогда, чтобы выполнить перестановку, необходимо действовать следующим образом для $0 \le j < d$ (параллельно, если это удобно). Присвоить $t \leftarrow x_j$ и $k \leftarrow j$. Затем, до тех пор, пока $(k+m) \bmod N \ne j$, присваивать $x_k \leftarrow x_{(k+m) \bmod N}$ и $k \leftarrow (k+m) \bmod N$. И наконец присвоить $x_k \leftarrow t$. В этом алгоритме неравенство $(k+m) \bmod N \ne j$ будет выполняться тогда и только тогда, когда $(k+m) \bmod N \ge d$, поэтому можно использовать любой тест, который является более эффективным. [W. Fletcher, R. Silver, *CACM* **9** (1966), 326.]

35. Положим $M = l + m + n$ и $N = l + 2m + n$. Циклы искомой перестановки получены из циклов такой перестановки на множестве $\{0, 1, \ldots, N-1\}$, которая переводит k в $(k + l + m) \bmod N$, просто исключая все элементы каждого цикла, которые $\ge M$. (Сравните это с аналогичной ситуацией в упр. 29.) *Доказательство.* Когда в результате предложенного в указании взаимного обмена будет выполнено присвоение $x_k \leftarrow x_{k'}$ и $x_{k'} \leftarrow x_{k''}$ для некоторого k, где $k' = (k + l + m) \bmod N$, $k'' = (k' + l + m) \bmod N$ и $k' \ge M$, получим, что $x_{k'} = x_{k''}$. Отсюда переход $\alpha\beta\gamma \to \gamma\beta\alpha$ заменяет x_k на $x_{k''}$.

Следовательно, существует ровно $d = \gcd(l+m, m+n)$ циклов и можно использовать алгоритм, аналогичный тому, который рассматривался в предыдущем упражнении.

Заслуживает также внимания несколько более простой способ сведения этой задачи к частному случаю из упр. 34, хотя он подразумевает больше случаев обращения к памяти. Предположим, $\gamma = \gamma'\gamma''$, где $|\gamma''| = |\alpha|$. Тогда можно заменить $\alpha\beta\gamma'\gamma''$ на $\gamma''\beta\gamma'\alpha$ и выполнить взаимный обмен γ'' с $\beta\gamma'$. Аналогичный подход можно использовать и при $|\alpha| > |\gamma|$. [См. J. L. Mohammed, C. S. Subi, *J. Algorithms* **8** (1987), 113–121.]

РАЗДЕЛ 1.4.1

1. Последовательность вызова: `JMP MAXN`; или `JMP MAX100`, если $n = 100$.
Состояния при входе: Для входа `MAXN` rI3 $= n$; предполагается, что $n \geq 1$.
Состояние при выходе: Такие же, как в (4).

2.
```
MAX50 STJ  EXIT
      ENT3 50
      JMP  2F
```

3. Состояние при входе: $n =$ rI1, если rI1 > 0; в противном случае $n = 1$.
Состояние при выходе: rA и rI2 такие же, как в (4); rI1 не изменился; rI3 $= \min(0, \text{rI1})$; rJ $=$ `EXIT` $+ 1$; CI не меняется, если $n = 1$, в противном случае CI будет больше, равен или меньше прежнего значения, в зависимости от того, будет ли максимум больше $X[1]$, равен $X[1]$ при rI2 > 1 или равен $X[1]$ при rI2 $= 1$.

(Аналогичное упражнение для (9), конечно, было бы немного более сложным.)

4.
```
SMAX1 ENT1 1      r = 1
SMAX  STJ  EXIT   Произвольное r.
      JMP  2F     Далее, как в прежней программе.
...
      DEC3 0,1    Уменьшить на r.
      J3P  1B
EXIT  JMP  *      Выход.
```

Последовательность вызова: `JMP SMAX`; или `JMP SMAX1`, если $r = 1$.
Состояние при входе: rI3 $= n$, предположительно положительному; для входа `SMAX` rI1 $= r$, предположительно положительному.
Состояние при выходе: rA $= \max_{0 \leq k < n/r}$ `CONTENTS(X` $+ n - kr$`)` $=$ `CONTENTS(X` $+$ rI2`)`; и rI3 $= (n-1) \bmod r + 1 - r = -((-n) \bmod r)$.

5. Можно использовать любой другой регистр, например:

Последовательность вызова:
```
ENTA *+2
JMP  MAX100.
```
Состояние при входе: Нет.
Состояние при выходе: Такие же, как в (4).

Код аналогичен (1), но первая команда имеет вид "`MAX100 STA EXIT(0:2)`".

6. (Решение предложено Джоэлом Голдбергом (Joel Goldberg) и Роджером М. Ааронсом (Roger M. Aarons).)

```
MOVE STJ  3F
     STA  4F          Сохранить rA и rI2.
     ST2  5F(0:2)
     LD2  3F(0:2)     rI2 ← адрес "NOP A,I(F)".
     LDA  0,2(0:3)    rA ← "A,I".
     STA  *+2(0:3)
```

```
        LD2  5F(0:2)    Восстановить rI2, так как в I может быть 2.
        ENTA *          rA ← индексированный адрес.
        LD2  3F(0:2)
        LD2N 0,2(4:4)   rI2 ← −F.
        J2Z  1F
        DECA 0,2
        STA  2F(0:2)
        DEC1 0,2        rI1 ← rI1 + F.
        ST1  6F(0:2)
2H      LDA  *,2
6H      STA  *,2
        INC2 1          Увеличивать rI2, пока значение в нем не станет равным нулю.
        J2N  2B
1H      LDA  4F         Восстановить rA и rI2.
5H      ENT2 *
3H      JMP  *          Выход к команде NOP.
4H      CON  0
```

7. (1) Операционная система распределяет высокоскоростную память (кэш-память) более эффективно, если программные блоки предназначены только для чтения. (2) Аппаратная кэш-память команды будет более быстрой и менее дорогой в случае, если команды не смогут измениться. (3) Рассмотрим (2), только с "каналом" вместо "кэш-памяти". Если команда модифицируется после входа в канал, то канал нужно очистить; схема проверки этого условия очень сложна, а ее использование отнимет много времени. (4) Самомодифицирующийся код может использоваться только одним процессом за один раз. (5) Самомодифицирующийся код может повредить программу отслеживания переходов (упр. 1.4.3.2–7), которая является важным инструментом диагностики, выполняющим "профилирование" (т. е. определяющим, сколько раз выполняется каждая команда).

РАЗДЕЛ 1.4.2

1. Если одна сопрограмма вызывает другую только один раз, то это всего лишь подпрограмма. Поэтому нужен такой пример, в котором каждая сопрограмма вызывает другую по меньшей мере в двух различных местах. Но даже в таком случае иногда можно установить что-то наподобие переключателя или воспользоваться каким-либо свойством данных, чтобы, входя в фиксированную точку внутри одной сопрограммы, можно было направляться к одному из двух нужных мест. И для этого опять-таки достаточно подпрограммы. А польза от применения сопрограмм становится все более очевидной по мере роста числа их обращений друг к другу.

2. Первый символ, найденный `IN`, будет потерян. [Сопрограмма `OUT` запускается первой, потому что в строках 58 и 59 выполняется необходимая инициализация `IN`. Чтобы первой запустить `IN`, нужно инициализировать `OUT` с помощью команды "`ENT4 -16`" и очистить буфер вывода, если неизвестно, пуст ли он. Затем можно из строки 62 перейти сначала в строку 39.]

3. Утверждение *почти* истинно, так как команда "`CMPA =10=`" внутри `IN` будет тогда единственным оператором сравнения в программе, а кодом "`.`" является 40. (!) Но если бы перед завершающей точкой стояло число повторений, то тест прошел бы, как по маслу. [*Замечание.* Если задаться целью создать эффективную программу и подойти к этому с максимальной педантичностью, то, вероятно, нужно будет удалить строки 40, 44 и 48 и вставить команду "`CMPA PERIOD`" между строками 26 и 27. Но если в сопрограммах используется флаг сравнения, то его необходимо занести в список характеристик сопрограммы в документации к программе.]

4. Приведем примеры трех различных компьютеров, имеющих историческое значение. (i) На машине IBM 650 на языке ассемблера SOAP мы написали бы последовательности вызова "LDD A" и "LDD B", а связь между сопрограммами выглядела бы так: "A STD BX AX" и "B STD AX BX" (предпочтительнее, чтобы две эти команды связи находились в оперативной памяти). (ii) На машине IBM 709, воспользовавшись обычными языками ассемблера, мы записали бы последовательность вызова в виде "TSX A,4" и "TSX B,4"; связь между сопрограммами выглядела бы так.

```
A   SXA  BX,4          B   SXA  AX,4
AX  AXT  1-A1,4        BX  AXT  1-B1,4
    TRA  1,4               TRA  1,4
```

(iii) На машине CDC 1604 последовательности вызова были бы "обратными переходами" (SLJ 4) к A или B, а команды связи между сопрограммами, например

```
A:  SLJ  B1;  ALS  0
B:  SLJ  A1;  SLJ  A'
```

были бы занесены в два последовательных 48-битовых слова.

5. "STA HOLDAIN; LDA HOLDAOUT" между OUT и OUTX и "STA HOLDAOUT; LDA HOLDAIN" между IN и INX.

6. Внутри сопрограммы A напишите команду "JMP AB", чтобы активизировать B, и команду "JMP AC" — чтобы активизировать C. Адреса BA, BC, CA, CB будут аналогичным образом использоваться и внутри B и C. Связь между сопрограммами будет иметь такой вид.

```
AB   STJ  AX        BC   STJ  BX        CA   STJ  CX
BX   JMP  B1        CX   JMP  C1        AX   JMP  A1
CB   STJ  CX        AC   STJ  AX        BA   STJ  BX
     JMP  BX             JMP  CX             JMP  AX
```

[*Замечание.* Если имеется n сопрограмм, то для осуществления связи подобного типа понадобится $2(n-1)n$ ячеек. Если n достаточно велико, то, конечно, можно использовать "централизованную" программу связи. Нетрудно придумать также метод, для которого нужно $3n+2$ ячеек. Но на практике для приведенного выше более быстрого метода требуется только $2m$ ячеек, где m — количество пар (i,j), таких, для которых сопрограмма i вызывает сопрограмму j. Когда существует много сопрограмм, каждая из которых независимо вызывает другие, то обычно используется внешняя последовательность операторов управления (более подробно этот вопрос обсуждается в разделе 2.2.5).]

РАЗДЕЛ 1.4.3.1

1. FCHECK используется только дважды, и оба раза сразу после нее вызывается подпрограмма MEMORY. Поэтому целесообразнее было бы создать для FCHECK специальный вход в подпрограмму MEMORY и сделать так, чтобы она помещала $-$R в rI2.

```
2. SHIFT J5N  ADDRERROR          DEC5 1
        DEC3 5                   J5P  2B
        J3P  FERROR              JMP  STOREAX
        LDA  AREG                SLA  1
        LDX  XREG                SRA  1
        LD1  1F,3(4:5)           SLAX 1
        ST1  2F(4:5)             SRAX 1
        J5Z  CYCLE               SLC  1
   2H   SLA  1              1H  SRC  1
```

```
3. MOVE  J3Z  CYCLE                    STX  0,1
         JMP  MEMORY                   LDA  CLOCK
         SRAX 5                        INCA 2
         LD1  I1REG                    STA  CLOCK
         LDA  SIGN1                    INC1 1
         JAP  *+3                      ST1  I1REG
         J1NZ MEMERROR                 INC5 1
         STZ  SIGN1(0:0)               DEC3 1
         CMP1 =BEGIN=                  JMP  MOVE  ▌
         JGE  MEMERROR
```

4. Просто вставьте "`IN 0(16)`" и "`JBUS *(16)`" между строками 003 и 004. (Разумеется, на другом компьютере все было бы совсем по-другому, поскольку нужно было бы выполнять преобразование в символьный код машины MIX.)

5. Общее время выполнения центральной управляющей программы составляет $34u$ плюс $15u$ на индексирование, если оно необходимо. Подпрограмма GETV работает $52u$ плюс $5u$, если $\mathrm{L} \neq 0$; дополнительное время, необходимое для загрузки, равно $11u$ для LDA или LDX, $13u$ — для LDi, $21u$ — для ENTA или ENTX, $23u$ — для ENTi (добавьте $2u$ к последним двум значениям времени, если $\mathrm{M} = 0$). Просуммировав, получим общее время $97u$ для LDA и $55u$ для ENTA плюс $15u$ на индексирование и плюс $5u$ или $2u$ при некоторых других условиях. Может показаться, что имитирование в данном случае дает для затраченного времени соотношение приблизительно 50:1. (В результате тестового прогона на $178u$ имитированного времени пришлось $8422u$ реального времени, т. е. получилось соотношение 47:1.)

7. При выполнении команд IN и OUT переменной, связанной с соответствующим устройством ввода-вывода, присваивается время, когда желательно выполнить передачу информации. Управляющая программа "CYCLE" опрашивает эти переменные в каждом цикле, чтобы узнать, не превысило ли значение CLOCK одну из них (или обе). Если превысило, то осуществляется передача информации и соответствующей переменной присваивается значение ∞. (Когда подобным образом нужно использовать более двух устройств ввода-вывода, переменных может оказаться так много, что лучше хранить их в рассортированном списке с помощью методов связи с памятью; см. раздел 2.2.5.) При имитации команды HLT необходимо очень осторожно завершать операции ввода-вывода.

8. Ложно. В rI6 может содержаться адрес ячейки BEGIN, если мы "попадем" в нее из ячейки BEGIN $-$ 1. Но тогда произойдет MEMERROR и будет предпринята попытка внести команду STZ в ячейку TIME! С другой стороны, всегда выполняется неравенство $0 \leq \mathrm{rI6} \leq$ BEGIN (это следует из строки 254).

РАЗДЕЛ 1.4.3.2

1. Замените строки 48 и 49 такой последовательностью команд.

```
XREG  ORIG *+2
LEAVE STX  XREG
      ST1  XREG+1
      LD1  JREG(0:2)
      LDA  -1,1
      LDX  1F
      STX  -1,1
            JMP  -1,1
1H          JMP  *+1
            STA  -1,1
            LD1  XREG+1
            LDX  XREG
            LDA  AREG
LEAVEX      JSJ  *
```

Особое значение здесь имеет, конечно, оператор "JSJ".

2.
```
* ПРОГРАММА ТРАССИРОВКИ
        ORIG *+99
BUF     CON  0
.............строки 02-04
        ST1  I1REG
.............строки 05-07
PTR     ENT1 -100
1H      JBUS *(0)
        STA  BUF+1,1(0:2)
.............строки 08-11
        STA  BUF+2,1
.............строки 12-13
        LDA  AREG
        STA  BUF+3,1
        LDA  I1REG
        STA  BUF+4,1
        ST2  BUF+5,1
        ST3  BUF+6,1
        ST4  BUF+7,1
        ST5  BUF+8,1
        ST6  BUF+9,1
        STX  BUF+10,1
        LDA  JREG(0:2)
        STA  BUF+1,1(4:5)
        ENTA 8
        JNOV 1F
        ADD  BIG
1H      JL   1F
        INCA 1
        JE   1F
        INCA 1
1H      STA  BUF+1,1(3:3)
        INC1 10
        J1N  1F
        OUT  BUF-99(0)
        ENT1 -100
1H      ST1  PTR(0:2)
.............строки 14-31
        LD1  I1REG
.............строки 32-35
        ST1  I1REG
.............строки 36-48
        LD1  I1REG
.............строки 49-50
B4      EQU  1(1:1)
BIG     CON  B4-8,B4-1(1:1)  ▌
```

После выполнения трассировки необходимо вызвать дополнительную программу, которая выводит на ленту содержимое последнего буфера и перематывает в начало устройство 0 (накопитель на магнитной ленте).

3. Запись на ленту происходит быстрее. Редактирование этой информации в текстовом виде во время трассировки потребовало бы слишком много памяти. Кроме того, *содержимое ленты можно напечатать выборочно.*

4. Истинная трассировка, которая необходима в упр. 6, не будет выполнена, так как нарушается ограничение (a), сформулированное в тексте раздела. Первая же попытка выполнить трассировку `CYCLE` приведет к зацикливанию и возврату к повторной трассировке `ENTER+1`, так как содержимое ячейки `PREG` испорчено.

6. Указание. Сохраняйте таблицу значений из всех ячеек памяти внутри области трассировки, которая была изменена внешней программой.

7. Программа трассировки должна просматривать программу, пока не найдет команду первого перехода (или условного перехода). После модификации этой и следующей команд она должна восстановить регистры и позволить программе выполнить все ее команды до данной точки "одним махом". [Этот метод может не пройти, если программа модифицирует собственные команды перехода или заменяет "непереходные" команды командами перехода. Для практических целей можно запретить подобные действия, сделав исключение только для команды `STJ`, которую, по всей видимости, так или иначе придется обрабатывать отдельно.]

РАЗДЕЛ 1.4.4

1. (a) Нет; операция ввода может оказаться незаконченной. (b) Нет: операция ввода может выполниться немного быстрее, чем команды `MOVE`. Поэтому данное предложение слишком рискованно.

2.
```
ENT1 2000
JBUS *(6)
MOVE 1000(50)
MOVE 1050(50)
OUT  2000(6)  ▌
```

```
3. WORDOUT STJ  1F                          DEC5 100
          STA  0,5                          JMP  2B
          INC5 1                    BUFMAX  CON  ENDBUF1
     2H   CMP5 BUFMAX               * BUFFER AREAS
     1H   JNE  *                    OUTBUF1 ORIG *+100
          OUT  -100,5(V)            ENDBUF1 CON  ENDBUF2
          LD5  0,5                  OUTBUF2 ORIG *+100
          ST5  BUFMAX               ENDBUF2 CON  ENDBUF1  ▮
```

В начале программы используйте команду "ENT5 OUTBUF1", а в конце программы, скажем,

```
          LDA  BUFMAX                       INC5 1
          DECA 100,5                        CMP5 BUFMAX
          JAZ  *+6                          JNE  *-3
          STZ  0,5                          OUT  -100,5(V)
```

4. Если время, затрачиваемое на вычисления, в точности равно времени В/В (это самая благоприятная ситуация), то одновременно работающие компьютер и периферийное устройство затрачивают в два раза меньше времени, чем в случае, когда они работают отдельно. Пусть C — время вычислений для всей программы, а T — общее время, требуемое для В/В. Тогда минимальное из возможных время выполнения при использовании буферизации составит $\max(C,T)$, а время выполнения без буферизации — $C+T$. И конечно, $\frac{1}{2}(C+T) \le \max(C,T) \le C+T$.

Но у некоторых устройств есть функция "штрафа за простаивание", которая становится причиной дополнительной потери времени, если между обращениями к устройству проходит слишком много времени. При этом с помощью буферизации можно получить более чем двойной выигрыш во времени (например, см. упр. 19).

5. Можно сократить время выполнения максимум в $(n+1)$ раз.

6. $\left\{\begin{array}{ll}\texttt{IN} & \texttt{INBUF1(U)}\\ \texttt{ENT6} & \texttt{INBUF2+99}\end{array}\right\}$ или $\left\{\begin{array}{ll}\texttt{IN} & \texttt{INBUF2(U)}\\ \texttt{ENT6} & \texttt{INBUF1+99}\end{array}\right\}$.

(Этим командам может предшествовать команда `IOC 0(U)`, перематывающая ленту только в случае необходимости).

7. Один из способов — использование сопрограмм.

```
INBUF1  ORIG *+100                          INC6 1
INBUF2  ORIG *+100                          J6N  2B
1H      LDA  INBUF2+100,6                   IN   INBUF1(U)
        JMP  MAIN                           ENN6 100
        INC6 1                              JMP  1B
        J6N  1B                   WORDIN    STJ  MAINX
WORDIN1 IN   INBUF2(U)            WORDINX   JMP  WORDIN1
        ENN6 100                  MAIN      STJ  WORDINX
2H      LDA  INBUF1+100,6         MAINX     JMP  *  ▮
        JMP  MAIN
```

Если добавить еще несколько команд, чтобы извлечь преимущества из частных случаев, эта программа будет работать быстрее, чем (4).

8. К моменту, показанному на рис. 23, два красных буфера уже заполнены образами строк и выводится на печать содержимое того буфера, на который указывает стрелка СЛЕДК. В это же время программа проводит вычисления по командам, расположенным между операциями ОСВОБОДИТЬ и НАЗНАЧИТЬ. Когда программа выполняет операции НАЗНАЧИТЬ, зеленый буфер, на который указывает стрелка СЛЕДЗ, становится желтым; стрелка СЛЕДЗ перемещается по часовой стрелке, и программа начинает заполнять желтый буфер. По окончании вывода стрелка СЛЕДК перемещается по часовой стрелке, буфер, содержимое которого только что было напечатано, становится зеленым и начинает печататься

содержимое оставшегося красного буфера. Наконец программа ОСВОБОЖДАЕТ желтый буфер и теперь он также готов к печати своего содержимого.

9, 10, 11.

Время	*Действие* ($N = 1$)	*Действие* ($N = 2$)	*Действие* ($N = 4$)
0	ASSIGN(BUF1)	ASSIGN(BUF1)	ASSIGN(BUF1)
1000	RELEASE, OUT BUF1	RELEASE, OUT BUF1	RELEASE, OUT BUF1
2000	ASSIGN (ожидать)	ASSIGN(BUF2)	ASSIGN(BUF2)
3000		RELEASE	RELEASE
4000		ASSIGN (ожидать)	ASSIGN(BUF3)
5000			RELEASE
6000			ASSIGN(BUF4)
7000			RELEASE
8000			ASSIGN (ожидать)
8500	BUF1 назначен, остановка вывода	BUF1 назначен, OUT BUF2	BUF1 назначен, OUT BUF2
9500	RELEASE, OUT BUF1	RELEASE	
10500	ASSIGN (ожидать)	ASSIGN (ожидать)	
15500			RELEASE

и т. д. Общее время при $N = 1$ равно $110000u$, при $N = 2$ — $89000u$, при $N = 3$ — $81500u$ и при $N \geq 4$ — $76000u$.

12. Замените последние три строки программы B следующими строками.

```
        STA  2F
        LDA  3F
        CMPA 15,5(5:5)
        LDA  2F
        LD5  -1,5
        DEC6 1
        JNE  1B
        JMP  COMPUTE
        JMP  *-1    (или JMP COMPUTEX)
    2H  CON  0
    3H  ALF  ␣␣␣␣.    ▮
```

13.
```
        JRED CONTROL(U)
        J6NZ *-1  ▮
```

14. Если $N = 1$, то алгоритм нарушается (может произойти обращение к буферу во время выполнения B/B); в противном случае результатом этой конструкции будет наличие двух желтых буферов. Это может оказаться полезным, если вычислительной программе понадобится обратиться к двум буферам одновременно, хотя это приводит к связыванию буферного пространства. В общем случае разность между количеством операций НАЗНАЧИТЬ и ОСВОБОДИТЬ должна быть неотрицательной и не превышать N.

15.
```
U       EQU  0
V       EQU  1
BUF1    ORIG *+100
BUF2    ORIG *+100
BUF3    ORIG *+100
TAPECPY IN   BUF1(U)
        ENT1 99
1H      IN   BUF2(U)
        OUT  BUF1(V)
        IN   BUF3(U)
        OUT  BUF2(V)
        IN   BUF1(U)
        OUT  BUF3(V)
        DEC1 3
        J1P  1B
        OUT  BUF1(V)
        HLT
        END  TAPECPY  ▮
```

Это частный случай алгоритма, показанного на рис. 26.

18. Частичное решение. В приведенных ниже алгоритмах t — это переменная, которая равна 0, когда устройство В/В свободно, и 1, когда оно активно.

Алгоритм А (НАЗНАЧИТЬ; *подпрограмма для нормального состояния*).

Этот алгоритм ничем не отличается от алгоритма 1.4.4А.

Алгоритм R (ОСВОБОДИТЬ; *подпрограмма для нормального состояния*).

R1. Увеличить n на единицу.

R2. Если $t = 0$, вызвать прерывание (с помощью команды INT) и в результате перейти к шагу В3. ▮

Алгоритм В (*Программа управления буферами, обрабатывающая прерывания*).

В1. Перезапустить главную программу.

В2. Если $n = 0$, присвоить $t \leftarrow 0$ и перейти к шагу В1.

В3. Присвоить $t \leftarrow 1$ и инициировать В/В из буферной области, на которую указывает стрелка СЛЕДК.

В4. Возобновить выполнение главной программы; выполнение условия "операция В/В завершена" приведет к прерыванию главной программы и переходу к шагу В5.

В5. Продвинуть СЛЕДК к следующему буферу по часовой стрелке.

В6. Уменьшить n на единицу и перейти к шагу В2. ▮

19. Если $C \le L$, то $t_k = (k-1)L$, $u_k = t_k + T$ и $v_k = u_k + C$ тогда и только тогда, когда $NL \ge T+C$. При $C > L$ ситуация усложняется; имеем $u_k = (k-1)C+T$ и $v_k = kC+T$ тогда и только тогда, когда существуют целые $a_1 \le a_2 \le \cdots \le a_n$, такие, что $t_k = (k-1)L + a_k P$ удовлетворяет неравенству $u_k - T \ge t_k \ge v_{k-N}$ для $N < k \le n$. Эквивалентным условием будет $NC \ge b_k$ при $N < k \le n$, где $b_k = C + T + ((k-1)(C-L)) \bmod P$. Пусть $c_l = \max\{b_{l+1}, \ldots, b_n, 0\}$; тогда последовательность c_l убывает и наименьшим значением N, при котором процесс остается стабильным, будет минимальное l, для которого $c_l/l \le C$. Так как $c_l < C + T + P$ и $c_l \le L + T + n(C-L)$, это значение N никогда не превышает $\lceil \min\{C+T+P, L+T+n(C-L)\}/C \rceil$. [См. A. Itai, Y. Raz, *CACM* **31** (1988), 1339–1342.]

РАЗДЕЛ 2.1

1. (a) SUIT(NEXT(TOP)) = SUIT(NEXT(242)) = SUIT(386) = 4. (b) Λ.

2. Всегда, когда V является переменной связи, значение которой не равно Λ (в противном случае выражение CONTENTS(V) не имеет смысла). Рекомендуется *избегать* использования LOC в таком контексте.

3. Пусть NEWCARD $\leftarrow$ TOP, и если TOP $\neq \Lambda$, то TOP $\leftarrow$ NEXT(TOP).

4. **С1.** Пусть X $\leftarrow$ LOC(TOP). (Для удобства можно сделать разумное предположение, что TOP $\equiv$ NEXT(LOC(TOP)), а именно — что значение TOP появляется в поле NEXT той ячейки, в которой оно хранится. Данное предположение относится и к программе (5); к тому же оно может избавить нас от необходимости создавать специальную процедуру для обработки случая, когда колода пуста.)

С2. Если NEXT(X) $\neq \Lambda$, то X $\leftarrow$ NEXT(X); затем повторить этот шаг.

С3. Пусть NEXT(X) $\leftarrow$ NEWCARD, NEXT(NEWCARD) $\leftarrow \Lambda$, TAG(NEWCARD) $\leftarrow 1$ ▮

5. **D1.** Пусть X $\leftarrow$ LOC(TOP), Y $\leftarrow$ TOP. (См. шаг С1. Согласно исходному предположению Y $\neq \Lambda$. Далее во всем алгоритме Y идет вслед за X в том смысле, что Y = NEXT(X).)

D2. Если NEXT(Y) $\neq \Lambda$, то X $\leftarrow$ Y, Y $\leftarrow$ NEXT(Y) Повторить этот шаг.

D3. (Теперь NEXT(Y) = Λ, поэтому Y указывает на последнюю (самую нижнюю) карту, а X — на предпоследнюю.) Пусть NEXT(X) $\leftarrow \Lambda$, NEWCARD $\leftarrow$ Y. ▮

6. Обозначения в пп (b) и (d), но *не* в п. (a)! `CARD` является узлом, а не связью с узлом.

7. Вариант (a) дает значение `NEXT(LOC(TOP))`, которое в данном случае идентично значению `TOP`; правильным является вариант (b). Повода для путаницы здесь нет Рассмотрим аналогичный пример, когда `X` является числовым значением: для занесения значения `X` в регистр A запишем `LDA X`, а не `ENTA X`, поскольку в последнем случае в регистр будет занесено значение `LOC(X)`.

8. Пусть rA ≡ `N`, rI1 ≡ `X`.

```
ENTA 0     B1. N ← 0.                    INCA 1          B3. N ← N + 1.
LD1  TOP       X ← TOP.                  LD1  0,1(NEXT)     X ← NEXT(X).
J1Z  *+4   B2. Верно ли, что X = Λ?      J1NZ *-2           ∎
```

9. Пусть rI2 ≡ `X`.

```
PRINTER EQU  18            Номер построчно печатающего устройства.
TAG     EQU  1:1
NEXT    EQU  4:5           Определение полей.
NAME    EQU  0:5
PBUF    ALF  PILE          Сообщение, которое печатается,
        ALF  EMPTY             если колода пуста.
        ORIG PBUF+24
BEGIN   LD2  TOP           Пусть X ← TOP.
        J2Z  2F            Колода пуста?
1H      LDA  0,2(TAG)      rA ← TAG(X).
        ENT1 PBUF          Приготовиться к выполнению команды MOVE.
        JBUS *(PRINTER)    Дождаться готовности принтера.
        JAZ  *+3           Верно ли, что TAG = 0
                               (обращена ли карта лицом вверх)?
        MOVE PAREN(3)      Нет· копировать скобки.
        JMP  *+2
        MOVE BLANKS(3)     Да: копировать пробелы.
        LDA  1,2(NAME)     rA ← NAME(X).
        STA  PBUF+1
        LD2  0,2(NEXT)     Пусть X ← NEXT(X).
2H      OUT  PBUF(PRINTER) Печатать строку.
        J2NZ 1B .          Если X ≠ Λ, повторить цикл печати.
DONE    HLT
PAREN   ALF      (
BLANKS  ALF
        ALF  )
        ALF                ∎
```

РАЗДЕЛ 2.2.1

1. Да. (Если элементы дека всегда будут вставляться с одного из двух его концов.)

2. *Для получения перестановки* 325641 *нужно выполнить последовательность действий* SSSXXSSXSXXX (согласно обозначениям из следующего упражнения). Перестановка 154623 не может быть получена, поскольку вагон 2 может предшествовать вагону 3 только в том случае, если он будет выведен из стека до вставки вагона 3.

3. Допустимой является такая последовательность, в которой количество действий X никогда не превышает количества действий S при их считывании слева направо.

Две разные допустимые последовательности действий должны давать разные результаты, так как если две последовательности совпадают вплоть до некоторой позиции, в которой одна последовательность содержит действие S, а другая — X, то последняя последовательность выведет некий символ, который в первой перестановке не может быть выведен из стека раньше символа, вставленного действием S из первой последовательности.

4. Эта задача эквивалентна многим другим интересным задачам, как, например, перечисление бинарных деревьев, подсчет количества способов вставки скобок в формулу, а также вычисление количества способов разбиения многоугольника на треугольники, и была впервые упомянута в 1759 году в записях Эйлера и фон Зегнера (см. раздел 2.3.4.6).

В приведенном ниже элегантном решении, принадлежащем А. Д. Андре (1878), используется принцип отражения (reflection principle). Очевидно, существует $\binom{2n}{n}$ последовательностей действий, которые содержат по n действий S и X. Остается только оценить количество *недопустимых* последовательностей (т. е. последовательностей, в которых содержится корректное соотношение S и X, но нарушаются другие условия). В любой недопустимой последовательности отметим первый символ X частичной последовательности до символа X включительно, в которой количество символов X превышает количество символов S. Тогда в этой частичной последовательности заменим каждый символ X символом S, а символ S — символом X. В результате получим последовательность с $(n+1)$ символами S и $(n-1)$ символами X. И наоборот, для каждой последовательности последнего типа этот процесс можно выполнить в обратном направлении и найти недопустимую последовательность первого типа, которая приводит к ней. Например, из последовательности SSXSXXXXXSSS таким образом можно получить последовательность XXSXSSSXXSSS. Благодаря такому соответствию получаем, что количество недопустимых последовательностей равно $\binom{2n}{n-1}$. Следовательно, $a_n = \binom{2n}{n} - \binom{2n}{n-1}$.

Используя эту же идею, можно решить более общую "задачу об оценке результатов баллотировки по выборочным данным" (ballot problem) из теории вероятностей, которая заключается в подсчете всех частичных недопустимых последовательностей для заданного количества символов S и X. Эта задача была решена еще в 1708 году Авраамом де Муавром, который показал, что количество последовательностей, содержащих l символов A и m символов B, а также по крайней мере одну исходную частичную строку, в которой символов A на n больше, чем символов B, равно $f(l,m,n) = \binom{l+m}{\min(m,l-n)}$. В частности, $a_n = \binom{2n}{n} - f(n,n,1)$, как показано выше. (Хотя де Муавр привел свой результат без доказательства [*Philos. Trans.* **27** (1711), 262–263], из контекста статьи ясно, что он знал доказательство, так как формула, очевидно, верна при $l \geq m+n$. Применяя его метод производящей функции для решения подобных задач, можно получить условие симметрии $f(l,m,n) = f(m+n, l-n, n)$ с помощью простых алгебраических выкладок.) Последующая история проблемы, связанной с оценкой результатов баллотировки по выборочным данным, и ее некоторые обобщения приведены в исчерпывающем обзоре D. E. Barton, C. L. Mallows, *Annals of Math. Statistics* **36** (1965), 236–260; см. также упр. 2.3.4.4–32 и раздел 5.1.4.

Теперь рассмотрим новый метод решения задачи об оценке результатов баллотировки по выборочным данным с помощью двойных производящих функций, поскольку этот метод подходит для решения более сложных задач, например такой, как в упр. 11.

Пусть g_{nm} — количество последовательностей действий S и X длиной n, в которых количество символов X превышает количество символов S, если считывать их слева направо, и в которых общее количество символов S на m больше, чем общее количество символов X. Тогда $a_n = g_{(2n)0}$. Очевидно, что g_{nm} равно нулю для четных значений $m+n$. Легко видеть, что для этих величин выполняются следующие рекуррентные соотношения:

$$g_{(n+1)m} = g_{n(m-1)} + g_{n(m+1)}, \quad m \geq 0, \quad n \geq 0; \qquad g_{0m} = \delta_{0m}.$$

Рассмотрим двойную производящую функцию $G(x,z)=\sum_{n,m} g_{nm}x^m z^n$ и допустим, что $g(z)=G(0,z)$. Тогда приведенное выше рекуррентное соотношение эквивалентно равенству

$$\left(x+\frac{1}{x}\right)G(x,z)=\frac{1}{x}g(z)+\frac{1}{z}(G(x,z)-1),\quad \text{т. е.}\quad G(x,z)=\frac{zg(z)-x}{z(x^2+1)-x}.$$

К сожалению, оно не дает ничего нового в случае, когда $x=0$. Несмотря на это попробуем разложить знаменатель на множители $z(1-r_1(z)x)(1-r_2(z)x)$, где

$$r_1(z)=\frac{1}{2z}(1+\sqrt{1-4z^2}),\quad r_2(z)=\frac{1}{2z}(1-\sqrt{1-4z^2}).$$

(Обратите внимание на то, что $r_1+r_2=1/z$; $r_1r_2=1$.) Продолжим эвристический анализ этого соотношения. Необходимо найти такое $g(z)$, чтобы для $G(x,z)$, заданного с помощью приведенной выше формулы, существовало бесконечное разложение в степенной ряд по x и z. Для функции $r_2(z)$ существует разложение в степенной ряд и $r_2(0)=0$. Более того, для фиксированного z при $x=r_2(z)$ знаменатель $G(x,z)$ стремится к нулю. Таким образом, необходимо выбрать величину $g(z)$ такой, чтобы числитель также стремился к нулю при $x=r_2(z)$. Иначе говоря, вероятно, следует выбрать $zg(z)=r_2(z)$. Благодаря сделанному выбору выражение для $G(x,z)$ упрощается:

$$G(x,z)=\frac{r_2(z)}{z(1-r_2(z)x)}=\sum_{n\ge 0}(r_2(z))^{n+1}x^n z^{-1}.$$

Именно такое разложение в степенной ряд удовлетворяет исходному равенству, а потому можно сделать вывод, что вид функции $g(z)$ был выбран верно.

Для решения данной задачи нужно найти коэффициенты $g(z)$. Действительно, в данном случае можно получить простое выражение для всех коэффициентов $G(x,z)$, применяя бином Ньютона:

$$r_2(z)=\sum_{k\ge 0} z^{2k+1}\binom{2k+1}{k}\frac{1}{2k+1}.$$

Пусть $w=z^2$ и $r_2(z)=zf(w)$. Тогда, применяя обозначения из упр. 1.2.6–25, получим $f(w)=\sum_{k\ge 0}A_k(1,-2)w^k$. Следовательно,

$$f(w)^r=\sum_{k\ge 0}A_k(r,-2)w^k.$$

Тогда

$$G(x,z)=\sum_{n,m}A_m(n,-2)\,x^n z^{2m+n},$$

и общее решение будет таким:

$$g_{(2n)(2m)}=\binom{2n+1}{n-m}\frac{2m+1}{2n+1}=\binom{2n}{n-m}-\binom{2n}{n-m-1};$$

$$g_{(2n+1)(2m+1)}=\binom{2n+2}{n-m}\frac{2m+2}{2n+2}=\binom{2n+1}{n-m}-\binom{2n+1}{n-m-1}.$$

5. Если $j<k$ и $p_j<p_k$, то элемент p_j нужно удалить из стека до размещения в нем элемента p_k. Если $p_j>p_k$, то элемент p_k должен оставаться в стеке до размещения в нем элемента p_j. Комбинируя эти два правила, получаем, что условие $p_j<p_k<p_i$ для $i<j<k$ *невыполнимо, так как оно означает, что элемент* p_j *нужно удалить из стека до элемента* p_k и после элемента p_i, однако p_i следует после p_k.

И наоборот, необходимая перестановка может быть получена с помощью следующего алгоритма: "Для $j = 1, 2, \ldots, n$ будем вставлять столько элементов, сколько требуется (ни одного или более), до тех пор, пока в стеке не появится элемент p_j. Затем удалим элемент p_j из стека". Этот алгоритм будет неверен, только если будет достигнуто значение j, для которого элемент p_j находится не на вершине стека, а покрыт некоторым другим элементом p_k, где $k > j$. Так как значения в стеке всегда монотонно возрастают, получим $p_j < p_k$. И элемент p_k мог быть там, если бы он был меньше p_i для некоторого $i < j$.

П. В. Раманан [*SICOMP* **13** (1984), 167–169] показал, как охарактеризовать перестановки, которые можно получить при работе со стеком при наличии m дополнительных ячеек памяти. (Это удивительно сложное обобщение данной задачи.)

6. По определению очереди можно получить только тривиальную перестановку $12 \ldots n$.

7. При работе с деком с ограниченным вводом для вывода первым элемента n необходимо вставить в очередь элементы $1, 2, \ldots, n$ во время первых n операций. А при работе с деком с ограниченным выводом для вывода первым элемента n во время первых n операций в очередь необходимо вставить элементы $p_1 p_2 \ldots p_n$. Следовательно, ответы будут такими: (a) 4132, (b) 4213, (c) 4231.

8. *При $n \leq 4$ не существует, а при $n = 5$ существует четыре такие перестановки* (см. упр. 13).

9. Выполняя с помощью дека с ограниченным выводом операции в обратном порядке для перестановки, полученной с помощью дека с ограниченным вводом, можно получить трижды обращенную перестановку. И наоборот, выполняя с помощью дека с ограниченным вводом операции в обратном порядке для перестановки, полученной с помощью дека с ограниченным выводом, можно также получить трижды обращенную перестановку. Это правило позволяет установить взаимно однозначное соответствие между двумя множествами перестановок.

10. (i) В допустимой последовательности должно быть n символов X и n символов S и Q вместе взятых. (ii) Количество символов X никогда не должно превышать общего количества символов S и Q при подсчете их слева направо. (iii) Всегда, когда количество символов X равняется общему количеству символов S и Q (при подсчете их слева направо), следующим должен быть символ Q. (iv) Две операции XQ никогда не должны выполняться одна за другой в таком порядке.

Необходимость правил (i) и (ii) очевидна. Дополнительные правила (iii) и (iv) добавлены во избежание двусмысленности, поскольку в пустой очереди операция S эквивалентна операции Q, а также потому, что последовательность действий XQ всегда можно заменить последовательностью QX. Таким образом, любая полученная последовательность соответствует по крайней мере одной допустимой последовательности.

Для того чтобы показать, как две разные допустимые последовательности приводят к разным перестановкам, рассмотрим последовательности, которые одинаковы вплоть до некоторой позиции. Например, в одной последовательности в этой позиции указано действие S, а в другой — действие X или Q. Согласно (iii), если очередь не пуста, в результате выполнения этих двух последовательностей действий будут получены разные перестановки (в отношении расположения элемента, вставленного при выполнении действия S). Другой возможный вариант заключается в том, что последовательности A и B одинаковы вплоть до некоторого действия, вслед за которым в A имеется действие Q, а в B — X. В таком случае в последовательности B после этого места также могут располагаться действия X, после которых согласно правилу (iv) не может быть символа Q, но согласно правилу (ii) должно следовать действие S. Поэтому перестановки будут разными.

11. По аналогии с упр. 4 допустим, что g_{nm} является количеством частичных допустимых последовательностей длиной n, которые оставляют m элементов в деке и *не* заканчиваются символом X. Аналогично определяется h_{nm}, но для тех последовательностей, которые *заканчиваются* символами X. Тогда $g_{(n+1)m} = 2g_{n(m-1)} + h_{n(m-1)}[m>1]$ и $h_{(n+1)m} = g_{n(m+1)} + h_{n(m+1)}$. Задав $G(x,z)$ и $H(x,z)$ по аналогии с определениями из упр. 4, получим

$$G(x,z) = xz + 2x^2z^2 + 4x^3z^3 + (8x^4+2x^2)z^4 + (16x^5+8x^3)z^5 + \cdots;$$
$$H(x,z) = z^2 + 2xz^3 + (4x^2+2)z^4 + (8x^3+6x)z^5 + \cdots.$$

Задав $h(z) = H(0,z)$, получим $z^{-1}G(x,z) = 2xG(x,z) + x\big(H(x,z)-h(z)\big) + x$, $z^{-1}H(x,z) = x^{-1}G(x,z) + x^{-1}\big(H(x,z)-h(z)\big)$ и, наконец,

$$G(x,z) = \frac{xz(x-z-xh(z))}{x-z-2x^2z+xz^2}.$$

Как и в упр. 4, попытаемся выбрать $h(z)$ таким, чтобы заменить часть числителя и знаменатель множителем в более простом виде. Например, $G(x,z) = xz/(1-2xr_2(z))$, где

$$r_2(z) = \frac{1}{4z}(z^2+1-\sqrt{(z^2+1)^2-8z^2}).$$

С учетом условия $b_0 = 1$ искомая производящая функция будет равна

$$\tfrac12(3-z-\sqrt{1-6z+z^2}) = 1+z+2z^2+6z^3+22z^4+90z^5+\cdots.$$

Выполнив дифференцирование, найдем удобное для вычислений рекуррентное соотношение: $nb_n = 3(2n-3)b_{n-1} - (n-3)b_{n-2}$, $n \ge 2$.

Другой способ решения этой задачи, предложенный В. Р. Праттом , заключается в использовании контекстно-свободных грамматик для набора строк (см. главу 10). Бесконечная грамматика с подстановками $S \to q^n(Bx)^n$, $B \to sq^n(Bx)^{n+1}B$ для всех $n \ge 0$ и $B \to \epsilon$ является однозначной и позволяет подсчитать количество строк с n x так же, как в упр. 2.3.4.4–31.

12. Согласно формуле Стирлинга $a_n = 4^n/\sqrt{\pi n^3} + O(4^n n^{-5/2})$. Для анализа b_n рассмотрим сначала общую задачу оценки коэффициента w^n в разложении в степенной ряд выражения $\sqrt{1-w}\,\sqrt{1-\alpha w}$, где $|\alpha| < 1$. Следовательно,

$$\sqrt{1-w}\,\sqrt{1-\alpha w} = \sqrt{1-w}\,\sqrt{1-\alpha+\alpha(1-w)} = \sqrt{1-\alpha}\sum_k \binom{1/2}{k}\beta^k(1-w)^{k+1/2},$$

где $\beta = \alpha/(1-\alpha)$, и искомый коэффициент равен $(-1)^n\sqrt{1-\alpha}\sum_k\binom{1/2}{k}\beta^k\binom{k+1/2}{n}$. Тогда

$$(-1)^n\binom{k+1/2}{n} = \binom{n-k-3/2}{n} = \frac{\Gamma(n-k-1/2)}{\Gamma(n+1)\Gamma(-k-1/2)} = -\frac{(k+1/2)^{\underline{k+1}}}{\sqrt{\pi}n}n^{\overline{-k-1/2}}$$

и $n^{\overline{-k-1/2}} = \sum_{j=0}^m \left[{-k-1/2 \atop -k-1/2-j}\right] n^{-k-1/2-j} + O(n^{-k-3/2-m})$ согласно 1.2.11.1–(16). Таким образом, получим асимптотическое разложение $[w^n]\sqrt{1-w}\sqrt{1-\alpha w} = c_0n^{-3/2} + c_1n^{-5/2} + \cdots + c_mn^{-m-3/2} + O(n^{-m-5/2})$, где

$$c_j = -\sqrt{\frac{1-\alpha}{\pi}}\sum_{k=0}^j\binom{1/2}{k}(k+\tfrac12)^{\underline{k+1}}\left\{{j+1/2 \atop k+1/2}\right\}\frac{\alpha^k}{(1-\alpha)^k}.$$

Для b_n запишем $1-6z+z^2 = (1-(3+\sqrt8)z)(1-(3-\sqrt8)z)$ и допустим, что $w = (3+\sqrt8)z$, $\alpha = (3-\sqrt8)/(3+\sqrt8)$. Тогда асимптотическая формула будет иметь следующий вид:

$$b_n = \frac{(\sqrt2-1)(3+\sqrt8)^n}{2^{3/4}\pi^{1/2}n^{3/2}}(1+O(n^{-1})) = \frac{(\sqrt2+1)^{2n-1}}{2^{3/4}\pi^{1/2}n^{3/2}}(1+O(n^{-1})).$$

13. В. Р. Пратт обнаружил, что перестановка невозможна тогда и только тогда, когда она содержит подпоследовательность с относительными значениями

$$5, 2, 7, 4, \ldots, 4k{+}1, 4k{-}2, 3, 4k, 1 \quad \text{или} \quad 5, 2, 7, 4, \ldots, 4k{+}3, 4k, 1, 4k{+}2, 3$$

для некоторого $k \geq 1$, либо такая же перестановка, но в которой поменяли местами два последних элемента или элементы 1 и 2, или одновременно и то, и другое. Таким образом, для $k = 1$ запрещенными являются следующие перестановки: 52341, 52314, 51342, 51324, 5274163, 5274136, 5174263, 5174236. [*STOC* 5 (1973), 268–277.]

14. (Решение предложено Р. Мелвиллом в 1980 году.) Пусть стеки R и S расположены так, что очередь начинается с верхнего элемента стека R, продолжается до нижнего элемента стека R, а затем проходит от нижнего элемента к верхнему элементу стека S. Если стек R пуст, то элементы стека S выталкиваются в стек R до полного опустошения стека S. Для удаления элемента из начала очереди следует вытолкнуть верхний элемент стека R, который не будет опустошен до тех пор, пока не будет пустой вся очередь в целом. Для вставки элемента с конца очереди следует протолкнуть элемент в стек S (если только стек R не пуст). Для извлечения из очереди элемент следует по крайней мере дважды вытолкнуть и дважды протолкнуть.

РАЗДЕЛ 2.2.2

1. `M` − 1 (*но не* `M`). Если допустить, что в очереди может находиться `M` элементов, как это предлагается в (6) и (7), то будет невозможно отличить пустую очередь от полной, проверяя только `R` и `F`, поскольку проверить можно будет только `M` состояний. Лучше пожертвовать одной ячейкой памяти, чем чрезмерно усложнить программу.

2. Вывод элемента с конца: если `R` = `F`, то `UNDERFLOW`, `Y` ← `X[R]`; если `R` = 1, то `R` ← `M`, в противном случае `R` ← `R` − 1. Ввод элемента с начала: пусть `X[F]` ← `Y`; если `F` = 1, то `F` ← `M`, в противном случае `F` ← `F` − 1; если `F` = `R`, то `OVERFLOW`.

3. (a) `LD1 I`; `LDA BASE,7:1`. Для этого потребуется 5 тактов вместо 4 или 8, как в (8).

(b) *Решение 1.* `LDA BASE,2:7`, причем для всех базовых адресов $I_1 = 0$, $I_2 = 1$. *Решение 2.* Если необходимо, чтобы для базовых адресов выполнялось условие $I_1 = I_2 = 0$, то можно записать `LDA X2,7:1`, где в ячейке `X2` содержится `NOP BASE,2:7`. Во втором решении потребуется выполнить на один такт больше, но в таком случае базовая таблица сможет использоваться с любыми значениями индексных регистров.

(c) Это эквивалентно `LD4 X(0:2)`, и для ее выполнения потребуется столько же времени, но регистр rI4 будет содержать значение +0, тогда как `X(0:2)` содержит −0.

4. (a) `NOP *,7` (b) `LDA X,7:7(0:2)`. (c) Это невозможно Код `LDA Y,7:7`, в котором по адресу `Y` содержится `NOP X,7:7`, нарушает ограничение, накладываемое на поле `7:7` (см. упр. 5). (d) `LDA X,7:1` с дополнительными константами

```
X  NOP  *+1,7:2
   NOP  *+1,7:3
   NOP  *+1,7:4
   NOP  0,5:6
```

Время выполнения равняется 6 тактам. (e) `INC6 X,7:6`, где `X` содержит `NOP 0,6:6`.

5. (a) Рассмотрим команду `ENTA 1000,7:7` с конфигурацией памяти

Ячейка	`ADDRESS`	I_1	I_2
1000.	1001	7	7
1001:	1004	7	1
1002:	1002	2	2

1003: 1001 1 1
1004: 1005 1 7
1005: 1006 1 7
1006: 1008 7 7
1007: 1002 7 1
1008: 1003 7 2

и с rI1 = 1, rI2 = 2. Найдем 1000,7,7 = 1001,7,7,7 = 1004,7,1,7,7 = 1005,1,7,1,7,7 = 1006,7,1,7,7 = 1008,7,7,1,7,7 = 1003,7,2,7,1,7,7 = 1001,1,1,2,7,1,7,7 = 1002,1,2,7,1,7,7 = 1003,2,7,1,7,7 = 1005,7,1,7,7 = 1006,1,7,1,7,7 = 1007,7,1,7,7 = 1002,7,1,1,7,7 = 1002,2,2,1,1,7,7 = 1004,2,1,1,7,7 = 1006,1,1,7,7 = 1007,1,7,7 = 1008,7,7 = 1003,7,2,7 = 1001,1,1,2,7 = 1002,1,2,7 = 1003,2,7 = 1005,7 = 1006,1,7 = 1007,7 = 1002,7,1 = 1002,2,2,1 = 1004,2,1 = 1006,1 = 1007. (Более быстрый способ выполнения этих выкладок вручную заключается в последовательной оценке адресов, указанных в позициях 1002, 1003, 1007, 1008, 1005, 1006, 1004, 1001, 1000 в этом порядке. Но компьютеру, очевидно, придется выполнить эту же оценку именно в заданном порядке.) Автор попытался применить несколько необычных схем изменения содержимого памяти во время оценки адреса с полным восстановлением исходного состояния после вычисления окончательного адреса. Аналогичные алгоритмы рассматриваются в разделе 2.3.5. Однако эти попытки были бесплодными и, похоже, для сохранения всей необходимой информации просто не хватит места.

(b, c) Пусть H и C являются вспомогательными регистрами, а N — счетчик. Для получения адреса M для команды в ячейке L необходимо сделать следующее.

A1. [Инициализация.] Установить $H \leftarrow 0$, $C \leftarrow L$, $N \leftarrow 0$. (В этом алгоритме C является "текущей" ячейкой, H используется для сложения содержимого различных индексных регистров, а N является мерой глубины косвенной адресации.)

A2. [Проверка адреса.] Установить $M \leftarrow \mathtt{ADDRESS}(C)$. Если $I_1(C) = j$, $1 \le j \le 6$, то $M \leftarrow M + rIj$. Если $I_2(C) = j$, $1 \le j \le 6$, то $H \leftarrow H + rIj$. Если $I_1(C) = I_2(C) = 7$, то $N \leftarrow N + 1$, $H \leftarrow 0$.

A3. [Косвенная адресация?] Если $I_1(C)$ или $I_2(C)$ равно 7, то $C \leftarrow M$, и перейти к шагу A2. В противном случае $M \leftarrow M + H$, $H \leftarrow 0$.

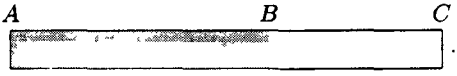
A4. [Сокращение глубины.] Если $N > 0$, то $C \leftarrow M$, $N \leftarrow N - 1$, и перейти к шагу A2. В противном случае M является искомым значением. ▮

Этот алгоритм можно корректно использовать в любой ситуации, за исключением случаев, когда $I_1 = 7$ и $1 \le I_2 \le 6$ *и* когда во время оценки адреса в поле ADDRESS возникает случай $I_1 = I_2 = 7$. Результат будет таков, как если бы значение I_2 было равно нулю. Для разъяснения принципа действия алгоритма A рассмотрим принятые в п. (a) обозначения. Состояние L,7,1,2,5,2,7,7,7,7 представляется с помощью C или M = L, N = 4 (количество замыкающих семерок) и H = rI1 + rI2 + rI5 + rI2 (постиндексирование). В решении для п. (b) этого упражнения значение счетчика N всегда будет равно либо 0, либо 1.

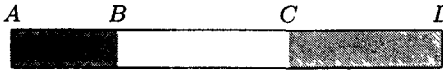
6. (c) приведет к переполнению (OVERFLOW), (e) приведет к недостатку (UNDERFLOW), а если выполнение программы будет продолжено, то в дальнейшем приведет к переполнению (OVERFLOW) для заключительного I_2.

7. Нет, так как TOP[i] должно быть больше, чем OLDTOP[i].

8. При работе со стеком полезная информация вставляется или удаляется с одной стороны, а свободная позиция — с другой:



Здесь $A = \texttt{BASE}[j]$, $B = \texttt{TOP}[j]$, $C = \texttt{BASE}[j+1]$. При работе с очередью или деком полезная информация вставляется или удаляется на концах, а свободная позиция в середине:



Или наоборот: информация вставляется или удаляется в середине, а свободное место — на концах:

Здесь $A = \texttt{BASE}[j]$, $B = \texttt{REAR}[j]$, $C = \texttt{FRONT}[j]$, $D = \texttt{BASE}[j+1]$. Для непустой очереди эти два случая отличаются условиями $B \le C$ и $B > C$ соответственно. Или, если известно, что очередь не переполнена, эти условия будут иметь вид $B < C$ и $B \ge C$ соответственно. Очевидно, что алгоритмы необходимо модифицировать таким образом, чтобы можно было расширять или сокращать пустые промежутки. (Таким образом, в случае переполнения, когда $B = C$, свободное пространство между B и C можно создать, перемещая только одну часть, а не обе сразу.) При вычислении $\texttt{SUM}$ и $\texttt{D}[j]$ на шаге G2 следует учесть, что каждая очередь занимает на одну ячейку больше, чем ей действительно требуется (см. упр. 1).

9. Для любой заданной последовательности действий $a_1, a_2, \ldots, a_m$ для каждой пары (j,k), где $j < k$ и $a_j > a_k$, необходима одна операция перемещения. (Такая пара называется инверсией (inversion); см. раздел 5.1.1.) Следовательно, количество таких пар и является количеством необходимых перемещений. Теперь представим все n^m записанных действий и для каждой из $\binom{m}{2}$ пар (j,k) с $j < k$ подсчитаем количество пар, для которых $a_j > a_k$. Очевидно, что оно равно $\binom{n}{2}$, т. е. количеству вариантов выбора a_j и a_k, умноженному на n^{m-2}, или количеству способов заполнения остальных мест. Значит, общее количество перемещений равно $\binom{m}{2}\binom{n}{2}n^{m-2}$. Для получения среднего количества перемещений, т. е. формулы (14), эту величину нужно разделить на n^m.

10. Используя такой же метод подсчета, как в упр. 9, получим

$$\binom{m}{2}\sum_{1\le j\le k\le n} p_j p_k = \frac{1}{2}\binom{m}{2}((p_1+\cdots+p_n)^2 - (p_1^2+\cdots+p_n^2))$$

$$= \frac{1}{2}\binom{m}{2}(1-(p_1^2+\cdots+p_n^2)).$$

В данной модели эта величина *абсолютно не зависит* от относительного расположения списков! (Поразмыслив, можно понять, почему это так: если рассмотреть все возможные перестановки заданной последовательности $a_1, \ldots, a_m$, можно обнаружить, что общее количество перемещений для всех перестановок зависит только от количества пар различных элементов $a_j \ne a_k$.)

11. Вычисляя среднее количество перемещений, как и прежде, получим

$$\frac{1}{n^m}\binom{n}{2}\sum_{0\le s<m}\sum_{r\ge t}\binom{s}{r}(n-1)^{s-r}n^{m-s-2}(m-s-1).$$

Здесь s представляет $j-1$ в обозначениях из предыдущего ответа, а r — количество элементов последовательности $a_1, a_2, \ldots, a_s$, которое равно a_j. Эту формулу можно упрощать с помощью ее производящих функций до тех пор, пока

$$\frac{1}{2n^t}\sum_{k=0}^{m-t}\binom{t-1+k}{k}\binom{m-t-k}{2}\left(1-\frac{1}{n}\right)^{k+1} \quad \text{для } t \ge 0.$$

Существует ли более простой способ решения этой задачи? По-видимому, нет, поскольку производящая функция для заданных n и t выглядит так.

$$\sum_m E_{mnt} z^m = \frac{n-1}{2n} \frac{z^2}{(1-z)^3} \Big(\frac{z}{n-(n-1)z} \Big)^t .$$

12. Если $m = 2k$, среднее значение равно 2^{-2k}, умноженному на

$$\binom{2k}{0} 2k + \binom{2k}{1}(2k-1) + \cdots + \binom{2k}{k} k + \binom{2k}{k+1}(k+1) + \cdots + \binom{2k}{2k} 2k.$$

И эта сумма равна

$$\binom{2k}{k} k + 2\Big(\binom{2k-1}{k} 2k + \cdots + \binom{2k-1}{2k-1} 2k \Big) = \binom{2k}{k} k + 4k \cdot \frac{1}{2} \cdot 2^{2k-1}.$$

Аналогичный аргумент можно использовать для $m = 2k+1$. Ответ в этом случае будет таким:

$$\frac{m}{2} + \frac{m}{2^m} \binom{m-1}{\lfloor m/2 \rfloor}.$$

13. А. Ч Яо доказал, что

$$\mathrm{E} \max(k_1, k_2) = \frac{1}{2} m + (2\pi(1-2p))^{-1/2} \sqrt{m} + O(m^{-1/2} (\log m)^2)$$

для больших m при $p < \frac{1}{2}$. [*SICOMP* **10** (1981), 398–403.] А Ф. П. М. Флажоле расширил этот анализ, в частности показав, что среднее количество асимптотически стремится к αm при $p = \frac{1}{2}$, где

$$\alpha = \frac{1}{2} + 8 \sum_{n \ge 1} \frac{\sin(n\pi/2) \cosh(n\pi/2)}{n^2 \pi^2 \sinh n\pi} \approx 0\ 67531\ 44833.$$

Более того, при $p > \frac{1}{2}$ окончательное распределение величины k_1 стремится к равномерному при $m \to \infty$, так что $\mathrm{E} \max(k_1, k_2) \approx \frac{3}{4} m$. [См. *Lecture Notes in Comp. Sci.* **233** (1986), 325–340.]

14. Пусть $k_j = n/m + \sqrt{n}\, x_j$. (Эта идея принадлежит Н. Г. де Брейну.) По формуле Стирлинга получим

$$\begin{aligned} n^{-m} \frac{m!}{k_1! \ldots k_n!} \max(k_1, \ldots, k_n) \\ = (\sqrt{2\pi})^{1-n} n^{n/2} \Big(\frac{m}{n} + \sqrt{m} \max(x_1, \ldots, x_n) \Big) \\ \times \exp\Big(-\frac{n}{2}(x_1^2 + \cdots + x_n^2) \Big) (\sqrt{m})^{1-m} \Big(1 + O\Big(\frac{1}{\sqrt{m}} \Big) \Big), \end{aligned}$$

где $k_1 + \cdots + k_n = m$ и значения x равномерно ограничены Их сумма по всем неотрицательным значениям $k_1, \ldots, k_n$, которые удовлетворяют этим условиям, является приближением интеграла Римана. А ее асимптотическое приближение будет равно $a_n(m/n) + c_n \sqrt{m} + O(1)$, где

$$a_n = (\sqrt{2\pi})^{1-n} n^{n/2} \int_{x_1 + \quad + x_n = 0} \exp\Big(-\frac{n}{2}(x_1^2 + \cdots + x_n^2) \Big)\, dx_2 \ldots dx_n,$$

$$c_n = (\sqrt{2\pi})^{1-n} n^{n/2} \int_{x_1 + \cdots + x_n = 0} \max(x_1, \ldots, x_n) \exp\Big(-\frac{n}{2}(x_1^2 + \cdots + x_n^2) \Big)\, dx_2 \ldots dx_n,$$

так как можно показать, что соответствующие суммы находятся в ϵ-окрестности величин a_n и c_n для любого ϵ.

Известно, что $a_n = \frac{1}{2}$, так как соответствующая сумма может быть оценена явно. Интеграл в выражении для c_n равен nI_1, где

$$I_1 = \int_{\substack{x_1+\cdots+x_n=0 \\ x_1\ge x_2,\ldots,x_n}} x_1 \exp\left(-\frac{n}{2}(x_1^2+\cdots+x_n^2)\right) dx_2\ldots dx_n.$$

Сделав подстановку

$$x_1 = \frac{1}{n}(y_2+\cdots+y_n), \quad x_2 = x_1 - y_2, \quad x_3 = x_1 - y_3, \quad \ldots, \quad x_n = x_1 - y_n,$$

можно найти $I_1 = I_2/n$, где

$$I_2 = \int_{y_2,\ldots,y_n\ge 0} (y_2+\cdots+y_n)\exp\Big(-\frac{Q}{2}\Big)\, dy_2\ldots dy_n,$$

и $Q = n(y_2^2+\cdots+y_n^2) - (y_2+\cdots+y_n)^2$. Теперь из соображений симметрии получим, что интеграл I_2 равен умноженному на $(n-1)$ тому же интегралу, в котором вместо $(y_2+\cdots+y_n)$ выполнена подстановка значений y_2. Следовательно, $I_2 = (n-1)I_3$, где

$$\begin{aligned} I_3 &= \int_{y_2,\ldots,y_n\ge 0} (ny_2 - (y_2+\cdots+y_n))\exp\Big(-\frac{Q}{2}\Big)\, dy_2\ldots dy_n \\ &= \int_{y_3,\ldots,y_n\ge 0} \exp\Big(-\frac{Q_0}{2}\Big)\, dy_3\ldots dy_n. \end{aligned}$$

Здесь Q_0 равно Q, где y_2 заменены нулями. [Если $n=2$, то $I_3 = 1$.] Теперь допустим, что $z_j = \sqrt{n}\,y_j - (y_3+\cdots+y_n)/(\sqrt{2}+\sqrt{n})$, $3\le j\le m$. Тогда $Q_0 = z_3^2+\cdots+z_n^2$, и можно вывести, что $I_3 = I_4/n^{(n-3)/2}\sqrt{2}$, где

$$\begin{aligned} I_4 &= \int_{y_3,\ldots,y_n\ge 0} \exp\left(-\frac{z_3^2+\cdots+z_n^2}{2}\right) dz_3\ldots dz_n \\ &= \alpha_n \int \exp\left(-\frac{z_3^2+\cdots+z_n^2}{2}\right) dz_3\ldots dz_n = \alpha_n(\sqrt{2\pi})^{n-2}. \end{aligned}$$

Здесь α_n — это отношение "телесного угла" в $(n-2)$-мерном пространстве, натянутом на векторы $(n+\sqrt{2n},0,\ldots,0) - (1,1,\ldots,1)$, $\ldots$, $(0,0,\ldots,n+\sqrt{2n}) - (1,1,\ldots,1)$, к полному телесному углу всего этого пространства. Следовательно,

$$c_n = \frac{(n-1)\sqrt{n}}{2\sqrt{\pi}}\alpha_n.$$

Получим $\alpha_2 = 1$, $\alpha_3 = \frac{1}{2}$, $\alpha_4 = \pi^{-1}\arctan\sqrt{2} \approx .304$ и

$$\alpha_5 = \frac{1}{8} + \frac{3}{4\pi}\arctan\frac{1}{\sqrt{8}} \approx .206.$$

[Хотя решение этой задачи получено для c_3 (Robert M. Kozelka , *Annals of Math. Stat.* **27** (1956), 507–512), для более высоких значений n оно, по-видимому, еще не опубликовано.]

16. Нет, если только на очереди не накладываются такие ограничения, при которых для них можно применить примитивный метод на основе правил (4) и (5).

17. Сначала следует показать, что всегда $\texttt{BASE}[j]_0 \le \texttt{BASE}[j]_1$. Затем можно заметить, что каждое переполнение стека i в последовательности $s_0(\sigma)$, которое необязательно является переполнением в последовательности $s_1(\sigma)$, происходит тогда, когда стек i становится

больше, чем когда-либо ранее, однако его новый размер не меньше, чем исходный размер, выделенный для стека i в последовательности $s_1(\sigma)$.

18. *Допустим, что затраты времени на выполнение одной операции вставки равны a плюс* $bN + cn$, если при этом необходимо выполнить переупаковку памяти, где N — количество занятых ячеек. А затраты на одну операцию удаления равны d. Предположим, что после переупаковки памяти, в результате которой N ячеек остаются занятыми и $S = M - N$ свободными, для каждой операции вставки до следующего перераспределения потребуется $a+b+10c+10(b+c)nN/S = O(1+n\alpha/(1-\alpha))$, где $\alpha = N/M$. Если p операций вставки и q операций удаления происходят до переупаковки памяти, то предполагаемые затраты равны $p(a+b+10c+10(b+c)nN/S)+qd$, тогда как реальные затраты составляют $pa+bN'+cn+qd \le pa + pb + bN + cn + qd$. Они меньше предполагаемых затрат, потому что $p > .1S/n$. Наше предположение о том, что $M \ge n^2$ означает, что $cS/n + (b+c)N \ge bN + cn$.

19. Можно было бы просто уменьшить все индексы на единицу, но приведенное ниже решение выглядит несколько элегантнее. В исходном состоянии $\mathtt{T} = \mathtt{F} = \mathtt{R} = 0$.

Протолкнуть Y в стек X: если $\mathtt{T} = \mathtt{M}$, то OVERFLOW; $\mathtt{X[T]} \leftarrow \mathtt{Y}$; $\mathtt{T} \leftarrow \mathtt{T}+1$.

Вытолкнуть Y из стека X: если $\mathtt{T} = 0$, то UNDERFLOW; $\mathtt{T} \leftarrow \mathtt{T}-1$; $\mathtt{Y} \leftarrow \mathtt{X[T]}$.

Вставить Y в очередь X: $\mathtt{X[R]} \leftarrow \mathtt{Y}$; $\mathtt{R} \leftarrow (\mathtt{R}+1) \bmod \mathtt{M}$; если $\mathtt{R} = \mathtt{F}$, то OVERFLOW.

Удалить Y из очереди X: если $\mathtt{F} = \mathtt{R}$, то UNDERFLOW; $\mathtt{Y} \leftarrow \mathtt{X[F]}$; $\mathtt{F} \leftarrow (\mathtt{F}+1) \bmod \mathtt{M}$.

Как и прежде, T — это количество элементов в стеке, а $(\mathtt{R}-\mathtt{F}) \bmod \mathtt{M}$ — количество элементов в очереди. Но самым верхним элементом стека теперь является X[T − 1], а не X[T].

Хотя специалистам в теории информатики почти всегда удобнее начинать отсчет с нуля, весь остальной мир вряд ли когда-нибудь перейдет к использованию нуль-индексирования. Что тут говорить, если даже Эдсгер Дейкстра при игре на фортепиано использует считалку "1–2–3–4 | 1–2–3–4"!

РАЗДЕЛ 2.2.3

1. Событие OVERFLOW неявно обрабатывается операцией $\mathtt{P} \Leftarrow \mathtt{AVAIL}$.

2.
```
INSERT  STJ  1F          Сохранить позицию NOP T.
        STJ  9F          Сохранить адрес выхода.
        LD1  AVAIL       rI1 ⇐ AVAIL.
        J1Z  OVERFLOW
        LD3  0,1(LINK)
        ST3  AVAIL
        STA  0,1(INFO)   INFO(rI1) ← Y.
1H      LD3  *(0:2)      rI3 ← LOC(T).
        LD2  0,3         rI2 ← T.
        ST2  0,1(LINK)   LINK(rI1) ← T.
        ST1  0,3         T ← rI1.
9H      JMP  *           ∎
```

3.
```
DELETE  STJ  1F          Сохранить позицию NOP T.
        STJ  9F          Сохранить адрес выхода.
1H      LD2  *(0:2)      rI2 ← LOC(T).
        LD3  0,2         rI3 ← T.
        J3Z  9F          Верно ли, что T = Λ?
        LD1  0,3(LINK)   rI1 ← LINK(T).
        ST1  0,2         T ← rI1.
        LDA  0,3(INFO)   rA ← INFO(rI1).
        LD2  AVAIL       AVAIL ⇐ rI3.
        ST2  0,3(LINK)
```

```
          ST3   AVAIL
          ENT3  2          Приготовиться ко второму выходу.
   9H     JMP   *,3          ▮
4. OVERFLOW STJ  9F         Сохранить значение регистра rJ.
          ST1   8F(0:2)     Сохранить значение rI1.
          LD1   POOLMAX
          ST1   AVAIL       Установить для AVAIL новый адрес.
          INC1  c
          ST1   POOLMAX     Увеличить POOLMAX.
          CMP1  SEQMIN
          JG    TOOBAD      Не исчерпан ли объем памяти?
          STZ   -c,1(LINK)  Установить LINK(AVAIL) ← Λ.
   9H     ENT1  *           Извлечь значение rJ.
          DEC1  2           Отнять 2.
          ST1   *+2(0:2)    Сохранить адрес выхода.
   8H     ENT1  *           Восстановить rI1.
          JMP   *           Возврат. ▮
```

5. Вставка с начала очереди очень похожа на основную операцию вставки (8) с дополнительной проверкой опустошения очереди: P ⇐ AVAIL, INFO(P) ← Y, LINK(P) ← F, если F = Λ, то R ← P; F ← P.

Для удаления элемента с конца очереди можно было бы найти узел, который связан с NODE(R), но данный метод крайне неэффективен, поскольку потребуется проследить весь путь от F. Это можно было бы выполнить так.

a) Если F = Λ, то UNDERFLOW, в противном случае установить P ← LOC(F).

b) Если LINK(P) ≠ R, то установить P ← LINK(P) и повторять этот этап до тех пор, пока LINK(P) станет равным R.

c) Установить Y ← INFO(R), AVAIL ⇐ R, R ← P, LINK(P) ← Λ.

6. Операцию LINK(P) ← Λ можно было бы удалить из (14), если удалить команды F ← LINK(P) и "если F = Λ, то R ← LOC(F)" из (17), а вместо последней из них вставить "если F = R, то F ← Λ и R ← LOC(F), в противном случае установить F ← LINK(P)".

В результате таких изменений поле LINK конечного узла очереди будет содержать бесполезную информацию, которая никогда не будет использована программой Благодаря подобной уловке можно сократить время выполнения программы, что очень полезно на практике. Однако при этом нарушаются основные допущения, принятые для сборки мусора (см. раздел 2.3.5), и ее нельзя использовать в таких алгоритмах.

7. (Убедитесь, что в вашем решении учитывается возможность опустошения списка)

I1. Установить P ← FIRST, Q ← Λ.

I2. Если P≠Λ, установить R←Q, Q←P, P←LINK(Q), LINK(Q) ←R и повторить этот шаг.

I3. Установить FIRST ← Q. ▮

По сути, все узлы сначала удаляются (выталкиваются) из одного стека, а затем вставляются (проталкиваются) в другой стек.

```
8.      LD1   FIRST      1   I1. P ≡ rI1 ← FIRST.
        ENT2  0          1   Q ≡ rI2 ← Λ.
        J1Z   2F         1   I2. Если список пуст, выполнить переход.
   1H   ENTA  0,2        n   R ≡ rA ← Q.
        ENT2  0,1        n   Q ← P.
        LD1   0,2(LINK)  n   P ← LINK(Q).
        STA   0,2(LINK)  n   LINK(Q) ← R.
```

```
     J1NZ 1B          n  P ≠ Λ?
2H   ST2  FIRST       1  I3. FIRST ← Q.  ▮
```

Время выполнения равно $(7n + 6)u$, но его можно сократить до $(5n + \text{constant})u$ (см. упр. 1.1–3).

9. (a) Да. (b) Да, если рассматривается биологическое отцовство (или материнство), и нет, если допускается юридическое отцовство (или материнство) (когда, например, дочь может выйти замуж за своего отца, совсем как в песне "I'm My Own Grampa"). (c) Нет ($-1 \prec 1$ и $1 \prec -1$). (d) Следует надеяться, что это так, ибо в противном случае возможно существование циклической цепочки доказательств. (e) $1 \prec 3$ и $3 \prec 1$. (f) Это утверждение не однозначно. Если предположить, что вызываемые y подпрограммы зависят от подпрограмм, которые вызвали y, то можно прийти к выводу о том, что условие транзитивности не всегда выполняется. (Например, общая подпрограмма ввода-вывода может вызывать различные процедуры обработки для каждого имеющегося устройства ввода-вывода, но не все эти процедуры обработки обычно необходимы для функционирования какой-то одной программы. Эта проблема часто является причиной ошибок во многих системах автоматического программирования.)

10. Для (i) существует три случая: $x = y$; $x \subset y$ и $y = z$; $x \subset y$ и $y \subset z$. А для (ii) — два случая: $x = y$; $x \neq y$. Решение очевидно для каждого из них, как и для (iii).

11. "Перемножьте" приведенные ниже комбинации, которые в итоге дадут 52 решения: $13749(25 + 52)86 + (1379 + 1397 + 1937 + 9137)(4258 + 4528 + 2458 + 5428 + 2548 + 5248 + 2584 + 5284)6 + (1392 + 1932 + 1923 + 9123 + 9132 + 9213)7(458 + 548 + 584)6$.

12. Рассмотрим следующие примеры. (a) Расположим (в любом порядке) все множества из k элементов до всех множеств с $k + 1$ элементами, $0 \leq k < n$. (b) Представим подмножество последовательностью нулей и единиц, которая указывает, какой элемент находится в множестве. Это позволит установить соответствие между всеми подмножествами и целыми числами от 0 до $2^n - 1$, которые записаны в двоичной системе счисления. Порядок соответствия и является топологической последовательностью.

13. Дж. Ша и Д. И. Кляйтман (*Discrete Math.* **63** (1987), 271–278) доказали, что это количество не превышает $\prod_{k=0}^{n} \binom{n}{k}^{\binom{n}{k}}$. Оно больше очевидной нижней границы $\prod_{k=0}^{n} \binom{n}{k}! = 2^{2^n(n+O(1))}$ на множитель $2^{2^n+O(n)}$, причем предполагается, что нижняя оценка ближе к истине.

14. Если $a_1\, a_2 \ldots a_n$ и $b_1 b_2 \ldots b_n$ — две допустимые топологические сортировки, то предположим, что j — такое минимальное значение, при котором $a_j \neq b_j$, тогда $a_k = b_j$ и $a_j = b_m$ для некоторого $k, m > j$. Теперь $b_j \not\preceq a_j$, поскольку $k > j$, а $a_j \not\preceq b_j$, так как $m > j$. Следовательно, (iv) не выполняется. И наоборот, если существует только одна топологическая сортировка $a_1\, a_2 \ldots a_n$, условие $a_j \preceq a_{j+1}$ должно выполняться для некоторого $1 \leq j < n$, так как в противном случае a_j и a_{j+1} можно поменять местами. Исходя из вышесказанного и условия транзитивности, получим (iv).

Замечание. Приведенное ниже альтернативное доказательство подходит и к бесконечным множествам. (a) Каждое частичное упорядочение может быть расширено до линейного упорядочения. Например, для некоторых двух элементов с отношениями $x_0 \not\preceq y_0$ и $y_0 \not\preceq x_0$ можно создать другое частичное упорядочение по правилу "$x \preceq y$, если $x \preceq x_0$ и $y_0 \preceq y$". Последнее упорядочение "включает" первое из двух упорядочений и $x_0 \preceq y_0$. Теперь для завершения доказательства применим обычным способом лемму Цорна или трансфинитную индукцию. (b) Очевидно, что линейное упорядочение не может быть расширено до любого другого линейного упорядочения. (c) Частичное упорядочение, которое имеет несравнимые элементы x_0 и y_0, как и в (a), может быть расширено до двух

линейных упорядочений, в которых $x_0 \preceq y_0$ и $y_0 \preceq x_0$ соответственно, а потому существует по крайней мере два линейных упорядочения.

15. Если S является конечным множеством, можно перечислить все отношения $a \prec b$ данного частичного упорядочения. Удаляя последовательно по одному все отношения, которые следуют из других, получим неприводимое множество. Задача теперь заключается в том, чтобы доказать единственность такого множества независимо от порядка удаления избыточных отношений. Если существует два таких неизбыточных множества U и V, в которых отношение $a \prec b$ присутствует в U, но не в V, то существует $k+1$ отношений $a \prec c_1 \prec \cdots \prec c_k \prec b$ в множестве V для некоторого $k \geq 1$. Однако в таком случае из множества U можно вывести отношения $a \prec c_1$ и $c_1 \prec b$ *без* использования отношения $a \prec b$ (так как $b \not\preceq c_1$ и $c_1 \not\preceq a$), поэтому *отношение* $a \prec b$ *является избыточным в множестве* U.

Такой результат не верен для бесконечных множеств S. Существует *не более* одного неизбыточного множества отношений. Например, если множество S включает целые числа плюс элемент ∞, а отношения $n \prec n+1$ и $n \prec \infty$ определены для всех n, то не существует *ни одного неизбыточного множества отношений, которые характеризуют это частичное* упорядочение.

16. Пусть $x_{p_1} x_{p_2} \ldots x_{p_n}$ — топологическая сортировка множества S. Тогда для строк и столбцов следует применить перестановку $p_1 p_2 \ldots p_n$.

17. Если k на шаге T4 увеличивается от 1 до n, то получим 1932745860. Если k на шаге T4 уменьшается от n до 1, как в программе T, то получим 9123745860.

18. Они связывают элементы списка в рассортированном порядке: `QLINK[0]` — первый, `QLINK[QLINK[0]]` — второй и т. д.; `QLINK[`последний`]` $= 0$.

19. В некоторых случаях это может привести к ошибке. Например, если в очереди содержится только один элемент на шаге T5, то модифицированный метод может установить `F` $= 0$ (опустошая таким образом очередь), но другие элементы могут быть помещены в очередь на шаге T6. Следовательно, в предлагаемой модификации этого алгоритма на шаге T6 необходимо включить проверку условия `F` $= 0$.

20. Действительно, стек *можно было бы* использовать следующим образом. (Шаг T7 исключается.)

T4. Установить `T` $\leftarrow 0$. Для $1 \leq k \leq n$, если `COUNT[`k`]` равно нулю, выполнить следующие действия: установить `SLINK[`k`]` $\leftarrow$ `T`, `T` $\leftarrow k$. (`SLINK[`k`]` $\equiv$ `QLINK[`k`]`.)

T5. Вывести значение `T`. Если `T` $= 0$, перейти к шагу T8, в противном случае установить `N` $\leftarrow$ `N` $- 1$, `P` $\leftarrow$ `TOP[T]`, `T` $\leftarrow$ `SLINK[T]`.

T6. То же, что и прежде, но перейти к шагу T5, а не к шагу T7. Когда `COUNT[SUC(P)]` уменьшится до нуля, установить `SLINK[SUC(P)]` $\leftarrow$ `T` и `T` $\leftarrow$ `SUC(P)`.

21. Повторяющиеся отношения могут лишь немного замедлить выполнение алгоритма и увеличить размер выделяемого в пуле пространства. Отношение $j \prec j$ будет рассматриваться как замкнутая петля (или цикл) (т. е. на соответствующей схеме это будет выглядеть, как выходящая из квадратика направленная на него же стрелка), которая нарушает частичное упорядочение.

22. Для создания "отказоустойчивой" программы следует: (a) проверить, что $0 < n <$ некоторое максимальное значение, (b) проверить, что для каждого отношения $j \prec k$ выполняются условия $0 < j, k \leq n$, (c) убедиться в том, что количество отношений не переполняет область пула.

23. В конце шага T5 добавьте `TOP[F]` $\leftarrow \Lambda$. (Затем всегда следует устанавливать `TOP[1]`, ..., `TOP[`n`]` для указания на все еще неотмененные отношения.) На шаге T8, если `N` > 0,

печатать LOOP DETECTED IN INPUT: и установить QLINK[k] $\leftarrow 0$ для $1 \le k \le n$. Кроме того, необходимо добавить следующие шаги.

T9. Для $1 \le k \le n$ установить P $\leftarrow$ TOP[k], TOP[k] $\leftarrow 0$ и выполнить шаг T10 (Это позволит установить указатель QLINK[j] на один из предшественников объекта j для каждого еще невыведенного j.) Затем перейти к шагу T11.

T10. Если P $\neq \Lambda$, установить QLINK[SUC(P)] $\leftarrow k$, P $\leftarrow$ NEXT(P) и повторить этот шаг.

T11. Найти k с QLINK[k] $\neq 0$.

T12. Установить TOP[k] $\leftarrow 1$ и $k \leftarrow$ QLINK[k]. Теперь, если TOP[k] $= 0$, повторить этот шаг.

T13. (Найдено начало петли.) Печатать значение k, установить TOP[k] $\leftarrow 0$, $k \leftarrow$ QLINK[k] и, если TOP[k] $= 1$, повторить этот шаг.

T14. Напечатать значение k (начало и конец петли) и прекратить выполнение. (*Замечание.* Петля распечатывается в обратном порядке; чтобы распечатать ее в прямом порядке, алгоритм, подобный описанному в упр. 7, следует вставить между шагами T12 и T13.) ▮

24. В код программы следует вставить следующие три новые строки.

```
08a PRINTER EQU  18
14a         ST6  NO
59a         STZ  X,1(TOP)          TOP[F] ← Λ.
```

Строки 74–75 следует заменить приведенными ниже строками.

```
 74         J6Z  DONE
 75         OUT  LINE1(PRINTER)    Печать сообщения о наличии петли.
 76         LD6  NO
 77         STZ  X,6(QLINK)        QLINK[k] ← 0.
 78         DEC6 1
 79         J6P  *-2               n ≥ k ≥ 1.
 80         LD6  NO
 81 T9      LD2  X,6(TOP)          P ← TOP[k].
 82         STZ  X,6(TOP)          TOP[k] ← 0.
 83         J2Z  T9A               Верно ли, что P = Λ?
 84 T10     LD1  0,2(SUC)          rI1 ← SUC(P).
 85         ST6  X,1(QLINK)        QLINK[rI1] ← k.
 86         LD2  0,2(NEXT)         P ← NEXT(P).
 87         J2P  T10               Верно ли, что P ≠ Λ?
 88 T9A     DEC6 1
 89         J6P  T9                n ≥ k ≥ 1.
 90 T11     INC6 1
 91         LDA  X,6(QLINK)
 92         JAZ  *-2               Найти такое k, для которого QLINK[k] ≠ 0.
 93 T12     ST6  X,6(TOP)          TOP[k] ← k.
 94         LD6  X,6(QLINK)        k ← QLINK[k].
 95         LD1  X,6(TOP)
 96         J1Z  T12               Верно ли, что TOP[k] = 0?
 97 T13     ENTA 0,6
 98         CHAR                   Преобразовать число k в символ.
 99         JBUS *(PRINTER)
100         STX  VALUE             Печатать.
101         OUT  LINE2(PRINTER)
```

```
102          J1Z  DONE          Остановиться, если TOP[k] = 0.
103          STZ  X,6(TOP)      TOP[k] ← 0.
104          LD6  X,6(QLINK)    k ← QLINK[k].
105          LD1  X,6(TOP)
106          JMP  T13
107  LINE1   ALF  LOOP          Строка заголовка.
108          ALF  DETEC
109          ALF  TED I
110          ALF  N INP
111          ALF  UT:
112  LINE2   ALF                Последующие строки.
113  VALUE   EQU  LINE2+3
114          ORIG LINE2+24
115  DONE    HLT                Конец вычислений.
116  X       END  TOPSORT                ▮
```

Замечание. Если отношения $9 \prec 1$ и $6 \prec 9$ добавить к исходным данным (18), то эта программа найдет петлю и напечатает ее в виде 9, 6, 8, 5, 9.

26. Одно из решений может включать две следующие стадии.

Стадия 1. (Таблица X используется, как последовательный стек, по мере того как применяются обозначения B = 1 или 2 для каждой используемой подпрограммы.)

A0. Для $1 \le J \le N$ установить $B(X[J]) \leftarrow B(X[J]) + 2$, если $B(X[J]) \le 0$.

A1. Если $N = 0$, перейти к стадии 2; в противном случае установить $P \leftarrow X[N]$ и уменьшить N на 1.

A2. Если $|B(P)| = 1$, перейти к шагу A1; в противном случае установить $P \leftarrow P + 1$.

A3. Если $B(SUB1(P)) \le 0$, установить $N \leftarrow N + 1$, $B(SUB1(P)) \leftarrow B(SUB1(P)) + 2$, $X[N] \leftarrow SUB1(P)$. Если $SUB2(P) \neq 0$ и $B(SUB2(P)) \le 0$, выполнить аналогичные действия с SUB2(P). Перейти к шагу A2. ▮

Стадия 2. (Проход по таблице и перераспределение памяти.)

B1. Установить $P \leftarrow FIRST$.

B2. Если $P = \Lambda$, установить $N \leftarrow N + 1$, $BASE(LOC(X[N])) \leftarrow MLOC$, $SUB(LOC(X[N])) \leftarrow 0$ и завершить выполнение алгоритма.

B3. Если $B(P) > 0$, установить $N \leftarrow N + 1$, $BASE(LOC(X[N])) \leftarrow MLOC$, $SUB(LOC(X[N])) \leftarrow P$, $MLOC \leftarrow MLOC + SPACE(P)$.

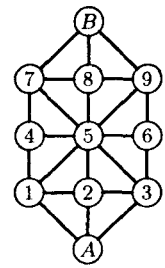
B4. Установить $P \leftarrow LINK(P)$ и вернуться к шагу B2. ▮

27. Читателю предлагается самостоятельно изучить приведенный ниже код.

```
B      EQU  0:1
SPACE  EQU  2:3
LINK   EQU  4:5
SUB1   EQU  2:3
SUB2   EQU  4:5
BASE   EQU  0:3
SUB    EQU  4:5
A0     LD2  N
       J2Z  B1
1H     LD3  X,2
       LDA  0,3(B)
       JAP  *+3
       INCA 2
       STA  0,3(B)
       DEC2 1
       J2P  1B
       LD1  N
A1     J1Z  B1
       LD2  X,1
       DEC1 1
A2     LDA  0,2(1:1)
       DECA 1
       JAZ  A1
       INC2 1
A3     LD3  0,2(SUB1)
       LDA  0,3(B)
       JAP  9F
       INC1 1
```

```
    INCA 2                    B1 ENT2 FIRST
    STA  0,3(B)                  LDA  MLOC
    ST3  X,1                     JMP  1F
 9H LD3  0,2(SUB2)            B3 LDX  0,2(B)
    J3Z  A2                      JXNP B4
    LDA  0,3(B)                  INC1 1
    JAP  A2                      ST2  X,1(SUB)
    INC1 1                       ADD  0,2(SPACE)
    INCA 2                    1H STA  X+1,1(BASE)
    STA  0,3(B)               B4 LD2  0,2(LINK)
    ST3  X,1                  B2 J2NZ B3
    JMP  A2                      STZ  X+1,1(SUB)  ▮
```

28. Предложим здесь лишь несколько комментариев, имеющих отношение к "военной игре". Пусть A — игрок с тремя фишками, которые расположены в узлах A13, а B — другой игрок. В этой игре A должен "захватить" B, но если B сможет дважды вызвать повторение одного и того же состояния, он выиграет. Однако для того, чтобы не хранить сведения обо всей предыстории игры в виде набора всех состояний, следует изменить алгоритм. Отметим состояния 157–4, 789–B и 359–6, в которых очередь следующего хода принадлежит игроку B, как "проигрышные" и применим предложенный алгоритм. Теперь основная выигрышная стратегия для игрока A заключается в перемещении только в проигрышные для B состояния. Но игроку A также нужно учитывать некоторые моменты, чтобы предыдущие ходы не повторялись. В хорошей игровой программе для выбора одного из нескольких возможных выигрышных вариантов следует использовать генератор случайных чисел. Итак, очевидным методом решения этой задачи с выигрышем для игрока A является всего лишь случайный перебор вариантов, которые ведут к проигрышному для игрока B состоянию Однако существует несколько интересных ситуаций, которые могут эту, на первый взгляд, благовидную процедуру привести к неудачному исходу! Рассмотрим, например, состояние 258–7 со следующим ходом игрока A, которое является выигрышным. Из состояния 258–7 игрок A может попытаться выполнить ход в состояние 158–7 (которое согласно этому алгоритму является проигрышным для игрока B). Но если игрок B сделает ход 158–B и вынудит игрока A сыграть 258–B, после которого игрок B снова сыграет 258–7, то игрок B выиграет, так как повторилось предыдущее состояние! В этом примере показано, что алгоритм должен повторно вызываться после каждого хода, начиная с каждого состояния, которое прежде было отмечено как "проигрышное" (если ход предстоит сделать игроку A) или "выигрышное" (если ход предстоит сделать игроку B). "Военная игра" вполне подходит для создания удовлетворительной демонстрационной программы.



Доска для "военной игры".

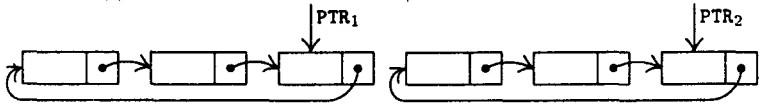
29. (a) Если FIRST = Λ, то ничего делать не надо, в противном случае установите P ← FIRST, а затем несколько раз (если вообще это потребуется) повторно устанавливайте P ← LINK(P) до тех пор, пока LINK(P) = Λ. Наконец установите LINK(P) ← AVAIL и AVAIL ← FIRST (и, вероятно, также FIRST ← Λ). (b) Если F = Λ, ничего не делать; в противном случае LINK(R) ← AVAIL и AVAIL ← F (и, вероятно, также F ← Λ, R ← LOC(F)).

30. Для вставки следует установить P ⇐ AVAIL, INFO(P) ← Y, LINK(P) ← Λ; если F = Λ, то F ← P, иначе — LINK(R) ← P и R ← P. Для удаления следует выполнить (9) с F вместо T. (Хотя неопределенное значение R удобно использовать для пустой очереди, это может привести к путанице в работе алгоритма сборки мусора из упр. 6.)

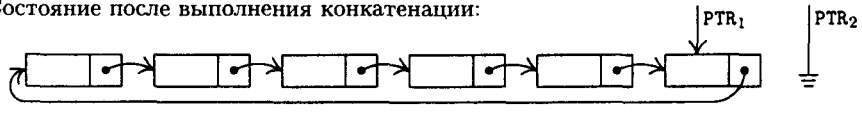
РАЗДЕЛ 2.2.4

1. Нисколько, во всяком случае даже затрудняет. (Указанное условие будет *не* совсем соответствовать идеологии циклических списков, если не допустить, что узел NODE(LOC(PTR)) является заголовком списка.)

2. Состояние до выполнения конкатенации:



Состояние после выполнения конкатенации:



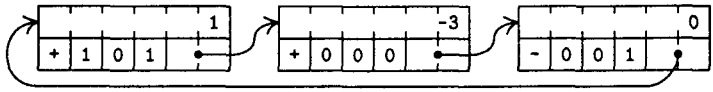
3. Если $PTR_1 = PTR_2$, единственным результатом операции будет $PTR_2 \leftarrow \Lambda$. Если $PTR_1 \neq PTR_2$, то обмен значениями связей приведет к разбиению списка на две части точно так, как удаление двух разных точек из окружности приводит к ее разбиению на две дуги. Во второй части этой операции указатель PTR_1 будет направлен на циклический список из узлов, которые следовало бы пройти, если бы в исходном списке последовательность связей была направлена от PTR_1 к PTR_2.

4. Пусть HEAD является адресом заголовка списка. Для проталкивания элемента Y в стек необходимо выполнить следующие действия: установить P ⇐ AVAIL, INFO(P) ← Y, LINK(P) ← LINK(HEAD), LINK(HEAD) ← P. Для выталкивания из стека элемента данных Y необходимо выполнить следующие действия: если LINK(HEAD) = HEAD, то UNDERFLOW; в противном случае установить P ← LINK(HEAD), LINK(HEAD) ← LINK(P), Y ← INFO(P), AVAIL ⇐ P.

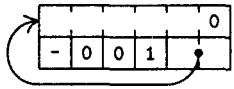
5. (Ср. с упр. 2.2.3–7.) Установить Q ← Λ, P ← PTR и повторять R ← Q, Q ← P, P ← LINK(Q), LINK(Q) ← R до тех пор, пока P ≠ Λ. (Затем установить Q = PTR.)

6.

(a)



(b)



7. При таком расположении поиск подобных членов может быть выполнен за счет одного прохода, а потому исключается необходимость повторного выполнения операций поиска произвольных элементов. Кроме того, расположение членов в порядке *возрастания* было бы несовместимо с меткой конца "−1".

8. При удалении узла или вставке нового узла перед ним необходимо знать, какой узел указывает на текущий узел. Однако для этого можно использовать и альтернативные способы. Узел NODE(Q) можно было бы удалить, задав Q2 ← LINK(Q) и NODE(Q) ← NODE(Q2), AVAIL ⇐ Q2. Узел NODE(Q2) можно было бы вставить перед NODE(Q), заменив значения NODE(Q2) ↔ NODE(Q) и задав LINK(Q) ← Q2, Q ← Q2. Благодаря таким искусным уловкам можно выполнить удаление и вставку, даже *не* зная, какой узел связан с узлом NODE(Q). Они использовались в ранних версиях языка IPL. Недостатком этого способа является то, что метка конца многочлена иногда может смещаться, а другие переменные связи могут быть связаны с ней.

9. Алгоритм A с P = Q просто удвоит многочлен (Q). Впрочем, как и следовало ожидать, за исключением случая, когда COEF = 0 для некоторого члена с ABC ≥ 0. Тогда он

не будет работать корректно. Алгоритм M с P = M также даст ожидаемый результат. Алгоритм M с P = Q приведет к тому, что многочлен (P) ← многочлен (P), умноженный на $(1+t_1)(1+t_2)\dots(1+t_k)$, если $M = t_1+t_2+\cdots+t_k$ (хотя это не так уж и очевидно). Если M = Q, алгоритм M также даст ожидаемый результат, многочлен (Q) ← многочлен (Q) + многочлен (Q) × многочлен (P), за исключением случая, когда постоянный член многочлена (P) равен −1 (тогда алгоритм не будет корректно работать).

10. Никакие (Единственное возможное отличие можно внести на шаге M2, а именно — удаление проверки переполнения каждого из полей A, B и C. Эти проверки не были указаны, поскольку предполагалось, что они не нужны.) Алгоритмы в этом разделе могут рассматриваться как операции над многочленами типа $f(x^{b^2}, x^b, x)$, а не многочленами типа $f(x, y, z)$.

11. Читателю предлагается самостоятельно изучить этот код.

```
COPY STJ  9F                    ST6  1,3(LINK)
     ENT3 9F                    ENT3 0,6
     LDA  1,1                   LD1  1,1(LINK)
1H   LD6  AVAIL                 LDA  1,1
     J6Z  OVERFLOW              JANN 1B
     LDX  1,6(LINK)             LD2  8F(LINK)
     STX  AVAIL                 ST2  1,3(LINK)
     STA  1,6                9H JMP  *
     LDA  0,1                8H CON  0  ▌
     STA  0,6
```

12. Пусть скопированный многочлен имеет p членов. Для выполнения программы A потребуется время, равное $(29p + 13)u$, причем для более корректного сравнения к нему следует добавить время создания нулевого многочлена, например $18u$ для программы из упр. 14. А для выполнения программы из упр. 11 потребуется время $(21p+31)u$, что почти в $\frac{3}{4}$ раза меньше.

13.

```
ERASE STJ  9F
      LDX  AVAIL
      LDA  1,1(LINK)
      STA  AVAIL
      STX  1,1(LINK)
9H    JMP  *  ▌
```

14.

```
ZERO STJ  9F                    MOVE 1F(2)
     LD1  AVAIL                 ST2  1,2(LINK)
     J1Z  OVERFLOW           9H JMP  *
     LDX  1,1(LINK)          1H CON  0
     STX  AVAIL                 CON  -1(ABC)  ▌
     ENT2 0,1
```

15.

```
MULT STJ  9F          Вход в подпрограмму.
     LDA  5F          Изменение значений переключателей.
     STA  SW1
     LDA  6F
     STA  SW2
     STA  SW3
     JMP  *+2
2H   JMP  ADD         M2. Цикл умножения.
1H   LD4  1,4(LINK)   M1. Следующий множитель. M ← LINK(M).
     LDA  1,4
```

```
     JANN  2B          Перейти к шагу M2, если ABC(M) ≥ 0.
8H   LDA   7F          Восстановить значения переключателей.
     STA   SW1
     LDA   8F
     STA   SW2
     STA   SW3
9H   JMP   *           Возврат.
5H   JMP   *+1         Новое значение переключателя SW1.
     LDA   0,1
     MUL   0,4         rX ← COEF(P) × COEF(M).
     LDA   1,1(ABC)    ABC(P)
     JAN   *+2
     ADD   1,4(ABC)      + ABC(M), если ABC(P) ≥ 0.
     SLA   2           Переместить в поле 0 3 из rA.
     STX   0F          Сохранить rX для SW2 и SW3.
     JMP   SW1+1
6H   LDA   0F          Новые значения SW2 и SW3.
7H   LDA   1,1         Обычное значение SW1.
8H   LDA   0,1         Обычное значение SW2 и SW3.
0H   CON   0           Временное хранилище
```

16. Пусть r — количество членов многочлена (M). Для выполнения этой подпрограммы потребуется $21pr+38r+29+27\sum m'+18\sum m''+27\sum p'+8\sum q'$ условных единиц времени. где суммирование проводится по соответствующим величинам во время r активизаций программы A. Количество членов многочлена (Q) возрастает на $p'-m'$ при каждой активизации программы A. Для вполне разумного предположения $m'=0$ и $p'=\alpha p$, где $0<\alpha<1$, получим, что соответствующие суммы равны 0, $(1-\alpha)pr$, αpr и $rq'_0+\alpha p(r(r-1)/2)$, где q'_0 есть значение q' во время первой итерации Общая сумма равна $4\alpha pr^2+40pr+4\alpha pr+8q'_0r+38r+29$. Из анализа следует, что в многочлене (M) желательно иметь меньше членов, чем в многочлене (P), так как при этом в многочлене (Q) придется чаще пропускать члены. которым нет подобных. (Более быстрый алгоритм описывается в упр. 5.2.3-29.)

17. На самом деле отличие невелико: операции сложения и умножения для списков любого типа будут практически одинаковы. По-видимому, преимущество циклического списка заметно проявится только при выполнении процедуры ERASE (см. упр. 13).

18. Пусть в поле связи узла x_i содержится значение $\mathtt{LOC}(x_{i+1})\oplus\mathtt{LOC}(x_{i-1})$, где символ "$\oplus$" обозначает операцию "исключающее или", хотя здесь может использоваться любая другая обратимая операция, например сложение или вычитание по модулю размера поля связи. Для упрощения процедуры запуска в циклическом списке удобно использовать два смежных заголовка списка. (Происхождение этого хитроумного метода остается неизвестным.)

РАЗДЕЛ 2.2.5

1. Вставить Y слева: P ⇐ AVAIL; INFO(P) ← Y; LLINK(P) ← Λ; RLINK(P) ← LEFT; если LEFT ≠ Λ, то LLINK(LEFT) ← P, в противном случае RIGHT ← P: LEFT ← P. Установить Y в крайнее слева положение и удалить: если LEFT = Λ, то UNDERFLOW; P ← LEFT; LEFT ← RLINK(P), если LEFT = Λ, то RIGHT ← Λ, в противном случае LLINK(LEFT) ← Λ, Y ← INFO(P); AVAIL ⇐ P

2. Рассмотрим случай с несколькими последовательными удалениями с одного конца. После каждого удаления необходимо знать, какой элемент нужно удалить следующим. Поэтому связи в списке должны быть направлены от этого конца списка. При удалении с

обоих концов списка предполагается, что связи должны быть направлены в обе стороны. В то же время в упр. 2.2.4–18 описывается, как можно представить две связи с помощью одного поля связи Поступив подобным образом, *можно* организовать такие действия для дека общего типа.

3. Чтобы показать независимость переменной CALLUP от переменной CALLDOWN, заметьте, к примеру, что в табл. 1 лифт не останавливается на этаже 2 или 3 в моменты 0393–0444, хотя там его ожидают люди. Они нажали кнопку CALLDOWN, но если бы они нажали кнопку CALLUP, лифт остановился бы там

Чтобы показать независимость переменной CALLCAR от других переменных, обратите внимание на то, что в табл. 1, когда двери начинают открываться в момент 1378, лифт уже принял решение о движении вверх (GOINGUP). Его состояние должно быть нейтральным (NEUTRAL) в этой точке, если CALLCAR[1] = CALLCAR[2] = CALLCAR[3] = CALLCAR[4] = 0 в соответствии с шагом E2, но на самом деле для переменных CALLCAR[2] и CALLCAR[3] задано значение 1 пассажирами лифта 7 и 9. (Если вообразить ту же ситуацию, в которой номера всех этажей увеличены на единицу, состояние STATE = NEUTRAL или STATE = GOINGUP при открытых дверях лифта приведет к тому, что лифт, вероятно, продолжит движение вниз либо безусловно двинется вверх.)

4. Если десять или более человек выходит на одном этаже, состояние может оставаться нейтральным все это время; и когда шаг E9 вызовет подпрограмму DECISION, это может привести к заданию нового состояния до того, как кто-то войдет в кабину лифта. На самом деле такое случается очень редко (и, несомненно, это было бы самое загадочное явление, которое автор наблюдал во время экспериментов с лифтом.)

5. Со времени открытия дверей лифта в момент 1063 и до момента 1183, когда человек 7 вошел в него, состояние лифта нейтрально, так как никаких вызовов с этажа 0 не поступило и в кабине лифта никого нет. Тогда пользователь 7 установит CALLCAR[2] $\leftarrow$ 1, и состояние лифта соответственно станет равным GOINGUP.

6. Нужно добавить условие "если OUT $<$ IN, то STATE $\neq$ GOINGUP; если OUT $>$ IN, то STATE $\neq$ GOINGDOWN" к условию "FLOOR = IN" на шагах U2 и U4. На шаге E4 следует взять пассажиров из очереди QUEUE[FLOOR], если они направляются в ту же сторону, что и лифт; при этом лифт не должен находиться в нейтральном состоянии STATE = NEUTRAL, иначе будут взяты все пассажиры.

[В здании факультета математики в Станфорде установлен именно такой лифт, но его пользователи на самом деле не обращают никакого внимания на внешние индикаторы, т. е. они входят в кабину независимо от направления движения. Почему бы конструкторам лифта не учесть этот факт и определить логику работы лифта без переменных CALLUP и CALLDOWN? Тогда работа лифта в целом ускорилась бы, так как он не останавливался бы так часто.]

7. В строке 227 предполагается, что этот человек находится в списке WAIT. Переход к шагу U4A подтверждает, что такое предположение является верным. Предполагается, что значение GIVEUPTIME положительно и действительно равно 100 или даже больше.

8. Читателю предлагается самостоятельно изучить этот код.

```
277  E8  DEC4 1
278      ENTA 61
279      JMP  HOLDC
280      LDA  CALL,4(3:5)
281      JAP  1F
282      ENT1 -2,4
283      J1Z  2F
```

```
284       LDA  CALL,4(1:1)
285       JAZ  E8
286  2H   LDA  CALL-1,4
287       ADD  CALL-2,4
288       ADD  CALL-3,4
289       ADD  CALL-4,4
290       JANZ E8
291  1H   ENTA 23
292       JMP  E2A
```

```
9. 01  DECISION STJ  9F              Сохранить адрес хранилища.
   02           J5NZ 9F              D1. Нужно ли принять решение?
   03           LDX  ELEV1+2(NEXTINST)
   04           DECX E1              D2. Следует открыть двери лифта?
   05           JXNZ 1F              Совершить переход,
   06           LDA  CALL+2                если лифт не в состоянии E1.
   07           ENT3 E3              Подготовиться к шагу E3,
   08           JANZ 8F                    если есть вызов на этаж 2.
   09  1H       ENT1 -4              D3. Есть ли вызовы?
   10           LDA  CALL+4,1        Поиск ненулевого значения
   11           JANZ 2F                    переменной вызова.
   12  1H       INC1 1               rI1 ≡ j − 4.
   13           J1NP *-3
   14           LDA  9F(0:2)         Равны нулю все CALL[j], j ≠ FLOOR.
   15           DECA E6B             Равен ли адрес выхода строке 250?
   16           JANZ 9F
   17           ENT1 -2              Установить j ← 2.
   18  2H       ENT5 4,1             D4. Изменение состояния.
   19           DEC5 0,4             STATE ← j − FLOOR.
   20           J5NZ *+2
   21           JANZ 1B              j = FLOOR не допускается в общем случае.
   22           JXNZ 9F              D5. Лифт в состоянии ожидания?
   23           J5Z  9F              Совершить переход, если лифт
                                           не на шаге E1 или j = 2.
   24           ENT3 E6              В противном случае запланировать E6.
   25  8H       ENTA 20              Ожидать в течение 20 единиц.
   26           ST6  8F(0:2)         Сохранить rI6.
   27           ENT6 ELEV1
   28           ST3  2,6(NEXTINST)   Установить NEXTINST для шага E3 или E6.
   29           JMP  HOLD            Запланировать действия.
   30  8H       ENT6 *               Восстановить rI6.
   31  9H       JMP  *               Выйти из подпрограммы. ∎
```

11. Пусть сначала LINK[k] $= 0$, $1 \le k \le n$, и HEAD $= -1$. На шаге изменения V[k] следует сообщить об ошибке, если LINK[k] $\neq 0$; в противном случае установить LINK[k] $\leftarrow$ HEAD, HEAD $\leftarrow k$ и определить для NEWV[k] новые значения V[k]. После каждого этапа моделирования установить $k \leftarrow$ HEAD, HEAD $\leftarrow -1$ и несколько раз (или ни разу) выполнять следующие действия до тех пор, пока не получим $k < 0$: установить V[k] $\leftarrow$ NEWV[k], $t \leftarrow$ LINK[k], LINK[k] $\leftarrow 0$, $k \leftarrow t$.

Ясно, что этот метод легко адаптируется для переменных, произвольным образом разбросанных в памяти, если включить поля NEWV и LINK в каждый узел переменной V.

12. В списке WAIT операции удаления выполняются слева направо, а операции вставки — справа налево (так как поиск будет, скорее всего, выполнен именно с этой стороны). Кроме того, *узлы удаляются из всех трех списков в нескольких местах, когда узел-предшественник* и узел-наследник удаляемого узла неизвестны. Только список ELEVATOR можно преобразовать в односвязный список без существенного снижения эффективности.

Замечание. В дискретном моделировании для сокращения времени сортировки было бы предпочтительнее применить нелинейный список типа WAIT. В разделе 5.2.3 рассматривается общая проблема обработки очередей по приоритету или списков наподобие "наименьший входит — первый выходит". Известно несколько способов, в которых для вставки или удаления в списке n элементов потребуется выполнить только $O(\log n)$ операций, хотя такой причудливый способ вряд ли стоит применять для малых n.

РАЗДЕЛ 2.2.6

1. (Здесь индексы находятся в диапазоне от 1 до n, а не от 0 до n, как в (6).) LOC(A[J,K]) = LOC(A[0,0]) + $2n$J + 2K, где предполагается, что A[0,0] на самом деле не существует. Если установить J=K=1, получим LOC(A[1,1]) = LOC(A[0,0]) + $2n$+2, а потому ответ можно представить несколькими способами. Тот факт, что значение LOC(A[0,0]) может быть отрицательным, часто приводит к появлению ошибок в работе компиляторов и процедур загрузки.

2. LOC(A[I_1,...,I_k]) = LOC(A[0,...,0]) + $\sum_{1\le r\le k} a_r \mathrm{I}_r$ = LOC(A[l_1,...,l_k]) + $\sum_{1\le r\le k} a_r \mathrm{I}_r - \sum_{1\le r\le k} a_r l_r$, где $a_r = c\prod_{r<s\le k}(u_s - l_s + 1)$.

Замечание. Обобщение для структур, которые встречаются в таких языках программирования, как C, и простой алгоритм для вычисления соответствующих констант можно найти в работе P. Deuel, *CACM* **9** (1966), 344–347.

3. $1 \le k \le j \le n$ тогда и только тогда, когда $0 \le k-1 \le j-1 \le n-1$. Поэтому заменим k, j, n значениями $k-1$, $j-1$, $n-1$ соответственно во всех формулах для нижней границы, равной нулю.

4. LOC(A[J,K]) = LOC(A[0,0]) + nJ − J(J − 1)/2 + K.

5. Пусть A0 = LOC(A[0,0]). Тогда существует по крайней мере два следующих решения (в которых предполагается, что J находится в регистре rI1, а K — в регистре rI2): (i) "LDA TA2,1:7", где по адресу TA2+j хранится команда "NOP j+1*j/2+A0,2"; (ii) "LDA C1,7:2", где по адресу C1 хранится команда "NOP TA,1:7", а по адресу TA+j — команда "NOP j+1*j/2+A0". В последнем случае потребуется выполнить на один цикл больше, но при этом таблица не будет связана с индексным регистром 2.

6. (a) $\mathrm{LOC(A[I,J,K])} = \mathrm{LOC(A[0,0,0])} + \binom{\mathrm{I}+2}{3} + \binom{\mathrm{J}+1}{2} + \binom{\mathrm{K}}{1}$.

(b) $\mathrm{LOC(B[I,J,K])} = \mathrm{LOC(B[0,0,0])} + \binom{n+3}{3} - \binom{n+3-\mathrm{I}}{3} + \binom{n+2-\mathrm{I}}{2} - \binom{n+2-\mathrm{J}}{2} + \mathrm{K} - \mathrm{J}$.

Следовательно, в данном случае можно получить аналогичное выражение для адреса произвольного элемента массива.

7. LOC(A[I_1,...,I_k]) = LOC(A[0,...,0]) + $\sum_{1\le r\le k} \binom{\mathrm{I}_r+k-r}{1+k-r}$. См. упр. 1.2.6–56.

8. (Решение П. Нэша.) Пусть элементы массива X[I,J,K] определены для $0 \le \mathrm{I} \le n$, $0 \le \mathrm{J} \le n+1$, $0 \le \mathrm{K} \le n+2$. Тогда A[I,J,K] = X[I,J,K]; B[I,J,K] = X[J,I+1,K]; C[I,J,K] = X[I,K,J+1]; D[I,J,K] = X[J,K,I+2]; E[I,J,K] = X[K,I+1,J+1]; F[I,J,K] = X[K,J+1,I+2]. Эта схема наиболее удобна, поскольку позволяет без накладок упаковать $(n+1)(n+2)(n+3)$ элементов шести тетраэдрических массивов в последовательном

порядке. *Доказательство.* A и B заполняют все ячейки X[i,j,k] с $k = \min(i,j,k)$; C и D заполняют все ячейки с $j = \min(i,j,k) \neq k$; E и F заполняют все ячейки $i = \min(i,j,k) \neq j,k$.

(Данную схему можно обобщить до m размерностей, если только потребуется упаковать $m!$ элементов *обобщенных тетраэдрических массивов с помощью* $(n+1) \times (n+2)\ldots(n+m)$ последовательных позиций. Для этого следует связать перестановку $a_1a_2\ldots a_m$ с каждым массивом и сохранить его элементы в массиве X[$I_{a_1} + B_1$, $I_{a_2} + B_2, \ldots, I_{a_m} + B_m$], где $B_1B_2\ldots B_m$ — таблица инверсии для $a_1a_2\ldots a_m$, которая определена в упр. 5.1.1–7.)

9. **G1.** Задать для P1, P2, P3, P4, P5, P6 значения, указывающие на первые ячейки списков FEMALE, A21, A22, A23, BLOND, BLUE соответственно. Положим в дальнейшем, что концом каждого списка является связь Λ, которая гораздо меньше любой другой связи. Если P6 = Λ, прекратить выполнение алгоритма (список, к сожалению, пуст).

G2. (Обход списков может выполняться по-разному, например сначала обрабатывается список EYES, затем — HAIR и AGE и наконец SEX.) Установить P5 ← HAIR(P5) несколько раз, пока не выполнится условие P5 ≤ P6 (если оно уже выполняется, это может и не потребоваться). Если теперь P5 < P6, перейти к шагу G5.

G3. Установить P4 ← AGE(P4) и повторять это действие до тех пор, пока не выполнится условие P4 ≤ P6. Аналогично выполнять те же действия для P3 и P2 до тех пор, пока не выполнятся условия P3 ≤ P6 и P2 ≤ P6. Если теперь P4, P3 и P2 меньше, чем P6, то перейти к шагу G5.

G4. Установить P1 ← SEX(P1) и повторять это действие до тех пор, пока не выполнится условие P1 ≤ P6. Если P1 = P6, значит, узел с заданным описанием девушки найден и можно вывести его адрес, P6. (Возраст можно определить с помощью указателей P2, P3 и P4.)

G5. Установить P6 ← EYES(P6). Прекратить выполнение алгоритма, если P6 = Λ; в противном случае вернуться к шагу G2. ▮

В этом алгоритме реализован интересный, но не лучший способ организации списка для выполнения подобного поиска.

10. См. раздел 6.5.

11. Не более $200 + 200 + 3 \cdot 4 \cdot 200 = 2800$ слов.

12. VAL(Q0) = c, VAL(P0) = b/a, VAL(P1) = d.

13. В поле, по которому упорядочен список, в конце каждого списка удобно иметь метку "меньший при сравнении". Простейший однонаправленный список *можно* было бы использовать, например, применив только связи LEFT в BASEROW[i] и связи UP в BASECOL[j] и изменив алгоритм S следующим образом. На шаге S2 проверить, верно ли, что P0 = Λ перед установкой J ← COL(P0), и, если верно, установить P0 ← LOC(BASEROW[I0]) и перейти к шагу S3. На шаге S3 проверить, верно ли, что Q0 = Λ, и, если верно, прекратить выполнение алгоритма. Шаг S4 следует изменить так же, как и шаг S2. На шаге S5 проверить, верно ли, что P1 = Λ, и, если верно, продолжить выполнение алгоритма, как если бы выполнялось условие COL(P1) < 0. На шаге S6 проверить, верно ли, что UP(PTR[J]) = Λ, и, если верно, продолжить выполнение алгоритма, как если бы значение поля ROW было отрицательным.

Эти изменения усложняют алгоритм и не способствуют экономии памяти за исключением поля ROW или COL в заголовках списков (а при работе с компьютером MIX это вообще не дает никакой экономии).

14. Сначала можно связать столбцы с ненулевыми элементами в осевой строке так, чтобы все остальные столбцы можно было пропустить в процессе преобразования строк. Строки, в которых значения осевого столбца равны нулю, пропускаются немедленно.

15. Пусть rI1 $\equiv$ PIVOT, J, rI2 $\equiv$ P0, rI3 $\equiv$ Q0, rI4 $\equiv$ P, rI5 $\equiv$ P1, X; LOC(BASEROW[i]) $\equiv$ BROW$+i$; LOC(BASECOL[j]) $\equiv$ BCOL $+ j$, PTR[j] $\equiv$ BCOL $+ j$(1:3).

```
01  ROW        EQU  0:3
02  UP         EQU  4:5
03  COL        EQU  0:3
04  LEFT       EQU  4:5
05  PTR        EQU  1:3
06  PIVOTSTEP  STJ  9F          Вход в подпрограмму, rI1 = PIVOT.
07  S1         LD2  0,1(ROW)    S1. Инициализация.
08             ST2  I0          I0 ← ROW(PIVOT).
09             LD3  1,1(COL)
10             ST3  J0          J0 ← COL(PIVOT).
11             LDA  =1.0=       Константа 1.
12             FDIV 2,1
13             STA  ALPHA       ALPHA ← 1/VAL(PIVOT).
14             LDA  =1.0=
15             STA  2,1         VAL(PIVOT) ← 1
16             ENT2 BROW,2      P0 ← LOC(BASEROW[I0]).
17             ENT3 BCOL,3      Q0 ← LOC(BASECOL[J0]).
18             JMP  S2
19  2H         ENTA BCOL,1
20             STA  BCOL,1(PTR) PTR[J] ← LOC(BASECOL[J]).
21             LDA  2,2
22             FMUL ALPHA
23             STA  2,2         VAL(P0) ← ALPHA × VAL(P0).
24  S2         LD2  1,2(LEFT)   S2. Обработка осевой строки. P0 ← LEFT(P0).
25             LD1  1,2(COL)    J ← COL(P0).
26             J1NN 2B          Если J ≥ 0, обработать J
27  S3         LD3  0,3(UP)     S3. Поиск новой строки. Q0 ← UP(Q0).
28             LD4  0,3(ROW)    rI4 ← ROW(Q0).
29  9H         J4N  *           Если rI4 < 0, выйти.
30             CMP4 I0
31             JE   S3          Если rI4 = I0, повторить.
32             ST4  I(ROW)      I ← rI4.
33             ENT4 BROW,4      P ← LOC(BASEROW[I]).
34  S4A        LD5  1,4(LEFT)   P1 ← LEFT(P)
35  S4         LD2  1,2(LEFT)   S4. Поиск нового столбца P0 ← LEFT(P0).
36             LD1  1,2(COL)    J ← COL(P0).
37             CMP1 J0
38             JE   S4          Если J = J0, повторить.
39             ENTA 0,1
40             SLA  2           rA(0 3) ← J.
41             J1NN S5
42             LDAN 2,3         Если J < 0,
43             FMUL ALPHA         установить VAL(Q0) ← −ALPHA × VAL(Q0).
44             STA  2,3
45             JMP  S3
```

```
46  1H      ENT4 0,5          P ← P1.
47          LD5  1,4(LEFT)    P1 ← LEFT(P).
48  S5      CMPA 1,5(COL)     S5. Поиск элемента I, J.
49          JL   1B           Выполнять цикл до тех пор, пока COL(P1) ≤ J.
50          JE   S7           Если =, перейти к шагу S7.
51  S6      LD5  BCOL,1(PTR)  S6. Вставка элемента I, J. rI5 ← PTR[J].
52          LDA  I            rA(0:3) ← I.
53  2H      ENT6 0,5          rI6 ← rI5.
54          LD5  0,6(UP)      rI5 ← UP(rI6).
55          CMPA 0,5(ROW)
56          JL   2B           Если ROW(rI5) > I, выполнить переход.
57          LD5  AVAIL        X ⇐ AVAIL.
58          J5Z  OVERFLOW
59          LDA  0,5(UP)
60          STA  AVAIL
61          LDA  0,6(UP)
62          STA  0,5(UP)      UP(X) ← UP(PTR[J]).
63          LDA  1,4(LEFT)
64          STA  1,5(LEFT)    LEFT(X) ← LEFT(P).
65          ST1  1,5(COL)     COL(X) ← J.
66          LDA  I(ROW)
67          STA  0,5(ROW)     ROW(X) ← I.
68          STZ  2,5          VAL(X) ← 0.
69          ST5  1,4(LEFT)    LEFT(P) ← X.
70          ST5  0,6(UP)      UP(PTR[J]) ← X.
71  S7      LDAN 2,3          S7. Осевая операция. −VAL(Q0)
72          FMUL 2,2              × VAL(P0)
73          FADD 2,5                + VAL(P1).
74          JAZ  S8           Если утрачен старший разряд, перейти к шагу S8.
75          STA  2,5          В противном случае сохранить в VAL(P1).
76          ST5  BCOL,1(PTR)  PTR[J] ← P1.
77          ENT4 0,5          P ← P1.
78          JMP  S4A          P1 ← LEFT(P), перейти к шагу S4.
79  S8      LD6  BCOL,1(PTR)  S8. Удаление элемента I, J. rI6 ← PTR[J].
80          JMP  *+2
81          LD6  0,6(UP)      rI6 ← UP(rI6).
82          LDA  0,6(UP)
83          DECA 0,5          Верно ли, что UP(rI6) = P1?
84          JANZ *-3          Выполнять цикл, пока не будет равно.
85          LDA  0,5(UP)
86          STA  0,6(UP)      UP(rI6) ← UP(P1).
87          LDA  1,5(LEFT)
88          STA  1,4(LEFT)    LEFT(P) ← LEFT(P1).
89          LDA  AVAIL        AVAIL ⇐ P1.
90          STA  0,5(UP)
91          ST5  AVAIL
92          JMP  S4A          P1 ← LEFT(P), перейти к шагу S4.  ▮
```

Замечание. Используя соглашения из главы 4, строки 71–74 можно переписать так:

```
LDA 2,3; FMUL 2,2; FCMP 2,5; JE S8; STA TEMP; LDA 2,5; FSUB TEMP
```

(с параметром EPSILON в ячейке памяти 0).

17. Чтобы получить строку i матрицы C, нужно сложить k-е строки матрицы B, умноженные на элементы $\mathtt{A}[i,k]$, такие, что $\mathtt{A}[i,k] \neq 0$. Для этого следует организовать связи между столбцами COL матрицы C, а связи ROW будут поддерживаться автоматически. [A. Schoor, *Inf. Proc. Letters* **15** (1982), 87–89.]

18. Трижды выполнив осевое преобразование в столбцах 3, 1, 2 соответственно, получим

$$\begin{pmatrix} \frac{1}{3} & \frac{2}{3} & \frac{1}{3} \\ -\frac{2}{3} & -\frac{1}{3} & -\frac{2}{3} \\ -\frac{1}{3} & -\frac{2}{3} & -\frac{1}{3} \end{pmatrix}, \quad \begin{pmatrix} \frac{1}{2} & \frac{1}{2} & 0 \\ -\frac{3}{2} & \frac{1}{2} & 1 \\ -\frac{1}{2} & -\frac{1}{2} & 0 \end{pmatrix}, \quad \begin{pmatrix} 0 & 1 & 0 \\ -2 & 1 & 1 \\ 1 & -2 & 0 \end{pmatrix}.$$

После заключительных перестановок получим следующий ответ:

$$\begin{pmatrix} 1 & -2 & 1 \\ 0 & 1 & -2 \\ 0 & 0 & 1 \end{pmatrix}.$$

20. $a_0 = \mathtt{LOC(A[1,1])} - 3$, $a_1 = 1$ либо 2, $a_2 = 3 - a_1$.

21. Например, $\mathtt{M} \leftarrow \max(\mathtt{I}, \mathtt{J})$, $\mathtt{LOC(A[I,J])} = \mathtt{LOC(A[1,1])} + \mathtt{M(M}-1) + \mathtt{I} - \mathtt{J}$. (Совершенно независимо было предложено сразу несколько таких формул. А. Л. Розенберг и Х. Р. Стронг предложили следующее k-мерное обобщение: $\mathtt{LOC(A[I}_1, \ldots, \mathtt{I}_k\mathtt{])} = \mathtt{L}_k$, где $\mathtt{L}_1 = \mathtt{LOC(A[1}, \ldots, \mathtt{1])} + \mathtt{I}_1 - 1$, $\mathtt{L}_r = \mathtt{L}_{r-1} + (\mathtt{M}_r - \mathtt{I}_r)(\mathtt{M}_r^{r-1} - (\mathtt{M}_r - 1)^{r-1})$ и $\mathtt{M}_r = \max(\mathtt{I}_1, \ldots, \mathtt{I}_r)$ [*IBM Tech. Disclosure Bull.* **14** (1972), 3026–3028]. Другие подобные результаты приводятся в *Current Trends in Programming Methodology* **4** (Prentice-Hall, 1978), 263–311.)

22. С помощью комбинаторных обозначений (упр. 1.2.6–56) можно записать

$$p(i_1, \ldots, i_k) = \binom{i_1}{1} + \binom{i_1 + i_2 + 1}{2} + \cdots + \binom{i_1 + i_2 + \cdots + i_k + k - 1}{k}.$$

[*Det Kongelige Norske Videnskabers Selskabs Forhandlinger* **34** (1961), 8–9.]

23. Пусть $c[\mathtt{J}] = \mathtt{LOC(A[0,J])} = \mathtt{LOC(A[0,0])} + m\mathtt{J}$, где m — количество строк в матрице в момент увеличения количества столбцов J на единицу; аналогично пусть $r[\mathtt{I}] = \mathtt{LOC(A[I,0])} = \mathtt{LOC(A[0,0])} + n\mathtt{I}$, где n — количество столбцов в матрице в момент увеличения количества строк I. Тогда функция размещения может иметь следующий вид:

$$\mathtt{LOC(A[I,J])} = \begin{cases} \mathtt{I} + c[\mathtt{J}], & \text{если } c[\mathtt{J}] \geq r[\mathtt{I}]; \\ \mathtt{J} + r[\mathtt{I}] & \text{в противном случае.} \end{cases}$$

Нетрудно доказать, что $c[\mathtt{J}] \geq r[\mathtt{I}]$ влечет $c[\mathtt{J}] \geq r[\mathtt{I}] + \mathtt{J}$, а $c[\mathtt{J}] \leq r[\mathtt{I}]$ влечет $c[\mathtt{J}] + \mathtt{I} \leq r[\mathtt{I}]$; следовательно, соотношение

$$\mathtt{LOC(A[I,J])} = \max(\mathtt{I} + \mathtt{LOC(A[0,J])}, \mathtt{J} + \mathtt{LOC(A[I,0])})$$

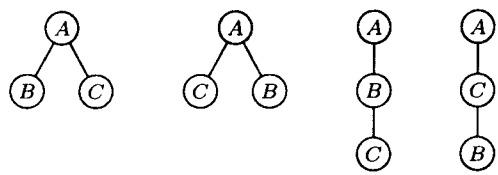
также будет выполняться. При этом необязательно накладывать ограничения на последовательное выделение mn ячеек. Единственное ограничение заключается в том, что при росте матрицы нужно последовательно расположить m или n новых ячеек по адресам, большим, чем адреса ранее использованных ячеек. Эта конструкция принадлежит Э. Дж. Оту и Т. Х. Мерретту [E. J. Otoo and T. H. Merrett, *Computing* **31** (1983), 1–9], которые также обобщили данный результат для k измерений.

24. [См. A. F. Aho, J. E. Hopcroft, and Ullman J. D., *The Design and Analysis of Computer Algorithms* (Addison-Wesley, 1974), exercise 2.12.] Помимо массива A, следует организовать проверочный массив V такого же размера и список L использованных ячеек памяти. Пусть n — количество элементов в списке L. В исходном состоянии $n = 0$ и содержимое L, A и V может быть произвольным. При каждой попытке доступа к элементу $\mathtt{A}[k]$ для k,

которое, может быть, еще не использовалось, сначала нужно проверить, выполняются ли условия $0 \le \mathtt{V}[k] < n$ и $\mathtt{L}[\mathtt{V}[k]] = k$. Если не выполняются, то необходимо установить $\mathtt{V}[k] \leftarrow n$, $\mathtt{L}[n] \leftarrow k$, $\mathtt{A}[k] \leftarrow 0$ и $n \leftarrow n+1$. В противном случае элемент $\mathtt{A}[k]$ наверняка содержит рабочие данные. (Несколько расширив этот метод, можно сохранить и впоследствии восстановить содержимое всех элементов A и V, которые изменяются во время этих вычислений.)

РАЗДЕЛ 2.3

1. Существует три способа выбора корня. После выбора в качестве корня, например, узла A существует еще три способа подразделения других узлов на поддеревья: $\{B\}$, $\{C\}$; $\{C\}$, $\{B\}$; $\{B, C\}$. В последнем случае существует также два способа превращения $\{B, C\}$ в дерево в зависимости от того, какой узел является корнем. Следовательно, получим четыре дерева с узлом A в качестве корня, а всего — 12 различных деревьев. Эта задача решается для любого количества узлов n в упр. 2.3.4.4–23.



2. Первые два дерева в ответе к упр. 1 идентичны, как ориентированные деревья, поэтому получим 9 различных возможностей. Общий случай рассматривается в разделе 2.3.4.4, в котором доказана справедливость формулы n^{n-1}.

3. Часть 1. Покажем, что существует *по крайней мере одна* такая последовательность. Пусть дерево содержит n узлов. Решение в случае, когда $n = 1$, очевидно, так как X является корнем. Если $n > 1$, согласно определению предполагается, что существует корень X_1 и поддеревья $T_1, T_2, \ldots, T_m$; либо $X = X_1$, либо X является членом какого-то единственного поддерева T_j. В последнем случае по индукции существует путь $X_2, \ldots, X$, где X_2 является корнем поддерева T_j, и, так как X_1 — родитель X_2, получим путь $X_1, X_2, \ldots, X$.

Часть 2. Покажем, что существует *не более одной* такой последовательности. Докажем по индукции, что, если X не является корнем дерева, X имеет единственного родителя (так что узел X_k определяет узел X_{k-1}, который определяет узел X_{k-2}, и т. д.). Если дерево имеет один узел, доказательство очевидно; иначе X находится в единственном поддереве T_j. Либо X является корнем T_j и по определению узел X имеет единственного родителя, либо X не является корнем T_j и в этом случае узел X по индукции имеет единственного родителя T_j, причем никакой другой узел за пределами поддерева T_j не может быть родителем узла X.

4. Верно (к сожалению).

5. 4.

6. Пусть родитель$^{[0]}(X) = X$ и пусть родитель$^{[k+1]}(X) =$ родитель(родитель$^{[k]}(X)$), так что родитель$^{[1]}(X)$ является родителем узла X, а родитель$^{[2]}(X)$ — прародителем узла X. Когда $k \ge 2$, родитель$^{[k]}(X)$ является пра$^{k-2}$прародителем X. Искомое условие родства заключается в том, что родитель$^{[m+1]}(X) =$ родитель$^{[m+n+1]}(Y)$, но родитель$^{[m]}(X) \ne$ родитель$^{[m+n]}(Y)$. Если $n > 0$, это условие несимметрично по отношению к X и Y, хотя оно считается симметричным в повседневных ситуациях.

7. Используйте несимметричное условие из упр. 6 и учтите тот факт, что родитель$^{[j]}(X) \ne$ родитель$^{[k]}(Y)$, если j или k (или оба сразу) равны -1. Чтобы показать, что это отношение всегда истинно для единственного значения m и единственного значения n, рассмотрим десятичную систему обозначений Дьюи для X и Y, а именно — $1.a_1.\cdots.a_p.b_1.\cdots.b_q$ и $1.a_1.\cdots.a_p.c_1.\cdots.c_r$, где $p \ge 0$, $q \ge 0$, $r \ge 0$ и $b_1 \ne c_1$ (если $qr \ne 0$).

В таком виде можно переписать обозначения для любой пары узлов, а потому, очевидно, следует принять, что $m = q - 1$ и $m + n = r - 1$

8. *Ни одно* бинарное дерево не является деревом Это совершенно разные понятия несмотря на то что схема непустого бинарного дерева очень похожа на дерево

9. Корнем является узел A, поскольку корень обычно располагается сверху

10. Любой *конечный* набор вложенных множеств можно сопоставить с лесом, определенным выше' Пусть A_1 , A_n — множества данного набора, которые не содержатся ни в каких других множествах Для фиксированного j совокупность всех множеств, которые содержатся в A_j, является вложенной, следовательно, можно допустить, что эта совокупность соответствует дереву, не упорядоченному с множеством A_j в качестве корня

11. Пусть *во вложенном наборе* $\mathcal{C}$ *отношение* $X \equiv Y$ *выполняется, если существует такое* $Z \in \mathcal{C}$ что $X \cup Y \subseteq Z$ Очевидно, что это отношение рефлексивно и симметрично, а также фактически является отношением эквивалентности, поскольку $W \equiv X$ и $X \equiv Y$ подразумевает, что существуют такие Z_1 и Z_2 в наборе $\mathcal{C}$ с $W \subseteq Z_1$, $X \subseteq Z_1 \cap Z_2$ и $Y \subseteq Z_2$ Так как $Z_1 \cap Z_2 \neq 0$, то либо $Z_1 \subseteq Z_2$, либо $Z_2 \subseteq Z_1$, следовательно, $W \cup Y \subseteq Z_1 \cup Z_2 \in \mathcal{C}$ Теперь, если $\mathcal{C}$ является вложенным набором, определим соответствующий набору $\mathcal{C}$ ориентированный лес согласно правилу "X является предком Y, а Y — потомком X тогда и только тогда, когда $X \supset Y$" Каждый класс эквивалентности набора $\mathcal{C}$ соответствует ориентированному дереву, которое является ориентированным лесом с отношением $X \equiv Y$ для всех X, Y (Таким образом, мы обобщили определения леса и дерева, которые ранее были даны только для конечных наборов) На основе этих рассуждений можно определить *уровень множества* X как кардинальное число множества предков (X) Аналогично *степень* множества X является кардинальным числом классов эквивалентности во вложенном наборе потомков (X) Назовем X *родителем* Y, а Y — *сыном* (*дочерью*) X, если X является предком Y, но не существует таких Z, что $X \supset Z \supset Y$ (X может иметь потомков, которые не являются его детьми, или иметь предков, которые не являются его родителями) Для *упорядочения* деревьев и леса некоторым специальным образом зададим порядок среди упомянутых выше классов эквивалентности, например, расширив отношение $\subseteq$ до понятия линейного порядка, как в упр 2 2 3–14

Пример (a) Пусть $S_{\alpha k} = \{ x \mid x = d_1 d_2 d_3$ в десятичной системе обозначений, где $\alpha = e_1 e_2 e_3$ в десятичной системе обозначений и $d_j = e_j$, если $j \bmod 2^k \neq 0\}$ Набор $\mathcal{C} = \{S_{\alpha k} \mid k \geq 0, 0 < \alpha < 1\}$ является вложенным и соответствует дереву с бесконечным количеством уровней и несчетной степенью каждого узла

Примеры (b) и (c) В таком случае удобно рассматривать это множество на плоскости, вместо того чтобы использовать действительные числа, к тому же между плоскостью и множеством действительных чисел существует взаимно однозначное соответствие Пусть $S_{\alpha mn} = \{(\alpha, y) \mid m/2^n \leq y < (m+1)/2^n\}$ и пусть $T_\alpha = \{(x, y) \mid x \leq \alpha\}$ Легко видеть, что набор $\mathcal{C} = \{S_{\alpha mn} \mid 0 < \alpha < 1, n \geq 0, 0 \leq m < 2^n\} \cup \{T_\alpha \mid 0 < \alpha < 1\}$ является вложенным Детьми множества $S_{\alpha mn}$ являются $S_{\alpha(2m)(n+1)}$ и $S_{\alpha(2m+1)(n+1)}$, а множество T_α имеет ребенка $S_{\alpha 00}$ плюс поддерево $\{S_{\beta mn} \mid \beta < \alpha\} \cup \{T_\beta \mid \beta < \alpha\}$ Таким образом, каждый узел обладает степенью 2 и имеет несчетное множество предков вида T_α Это построение предложено Р Г Байглоу (R Bigelow)

Замечание Данное построение можно усовершенствовать, если подходящим образом упорядочить множество действительных чисел и определить $T_\alpha = \{(x, y) \mid x \succ \alpha\}$, получая в результате вложенный набор, в котором каждый узел имеет несчетный уровень, степень 2 и двоих детей

12. Для того чтобы обеспечить соответствие леса частично упорядоченному множеству, наложим на него дополнительное ограничение (аналогично "вложенным множествам") *если* $x \preceq y$ *и* $x \preceq z$, *то либо* $y \preceq z$, *либо* $z \preceq y$ Иначе говоря, элементы, которые

больше данного элемента, являются линейно упорядоченными. Для получения дерева также следует предположить существование такого наибольшего элемента r, что $x \preceq r$ для всех x. Доказательство того факта, что в результате получится определенное выше неупорядоченное дерево, если число узлов конечно, проводится так же, как и для вложенных множеств в упр. 10.

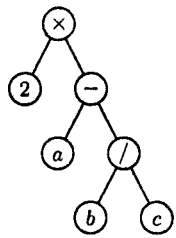
13. a_1, $a_1.a_2$, ..., $a_1.a_2.\cdots.a_k$.

14. Поскольку множество S не является пустым, оно содержит элемент $1\ a_1.\cdots.a_k$, где k принимает минимально возможное значение. Если $k > 0$, выберем в множестве S минимально возможное значение a_k и сразу же получим, что k должно быть равно 0. Иначе говоря, S должно содержать элемент 1. Пусть этот элемент является корнем. Все остальные элементы $k > 0$, а потому оставшиеся элементы множества S можно разбить на множества $S_j = \{1.j.a_2.\cdots.a_k\}$, $1 \le j \le m$, для некоторого $m \ge 0$. Если $m \ne 0$ и множество S_m не пусто, то по тем же соображениям получим, что $1.j$ принадлежит множеству S_j для каждого S_j. Поэтому каждое множество S_j не пусто. Тогда легко видеть, что множества $S'_j = \{1.a_2.\cdots.a_k \mid 1.j.a_2.\cdots.a_k$ принадлежит $S_1\}$ удовлетворяют тому же условию, что и множество S. По индукции каждое множество S_j образует дерево.

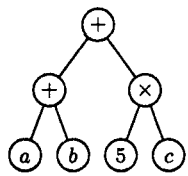
15. Пусть 1 — корень, а $\alpha.0$ и $\alpha.1$ — корни левого и правого поддеревьев дерева α соответственно, если такие корни существуют. Например, на рис. 18, (а) Король Кристиан IX встречается дважды, а именно — в позициях 1.0.0.0.0 и 1.1.0 0.1.0. Для краткости опустим десятичные точки в этих обозначениях и получим 10000 и 110010.

Замечание. Это обозначение предложено Ф. Гальтоном (Francis Galton); см. *Natural Inheritance* (Macmillan, 1889), 249. Для родословных лучше использовать мнемонические обозначения F (father — отец) и M (mother — мать) вместо 0 и 1 и отбросить начальную единицу. Тогда родственное отношение Кристиана IX по отношению к Чарльзу можно выразить обозначением *MFFMF*, т. е Кристиан IX является отцом матери отца отца матери Чарльза. Система обозначений на основе 0 и 1 интересна еще и тем, что она позволяет установить важное соотношение между узлами бинарного дерева и положительными целыми числами в двоичной системе (а именно — между адресами в памяти компьютера).

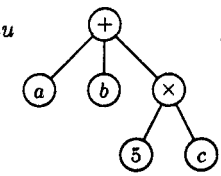
16. (a)



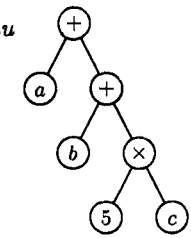
(b)



или



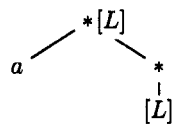
, *или*



17. Узел-родитель $(F[1]) = A$, узел-родитель $(F[1,2]) = C$; следовательно, узел-родитель $(F[1,2,2]) = E$.

18. $L[5,1,1] = (2)$. $L[3,1]$ не имеет смысла, так как $L[3]$ является пустым Списком.

19.



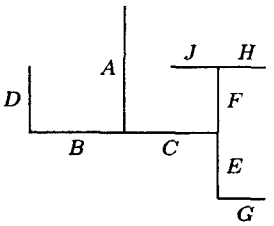
$L[2] = (L)$; $L[2,1,1] = a$.

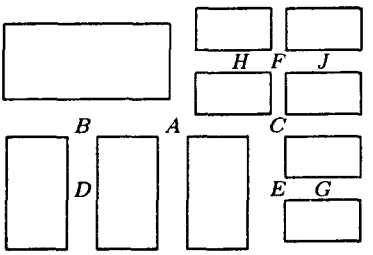
20. (Интуитивно соответствие между 0-2-деревьями и бинарными деревьями можно получить, удалив все концевые узлы в 0-2-дереве См. построение в разделе 2.3 4.5.) Пусть 0-2-дерево с одним узлом соответствует пустому бинарному дереву, а 0-2-дерево с более чем одним узлом, состоящее из корня r и 0-2-деревьев T_1 и T_2, соответствует бинарному дереву с корнем r, левым поддеревом T'_1 и правым поддеревом T'_2, где T_1 и T_2 соответствуют T'_1 и T'_2.

21. $1+0\cdot n_1+1\cdot n_2+\cdots+(m-1)\cdot n_m$. *Доказательство.* Количество узлов в дереве равно $n_0+n_1+n_2+\cdots+n_m$, и оно также равно $1+$ (количество детей в дереве) $=1+0\cdot n_0+1\cdot n_1+2\cdot n_2+\cdots+m\cdot n_m$.

22. Основная идея заключается в использовании рекурсивного построения, т. е. в представлении непустого бинарного дерева в виде корня, а также левого и правого поддеревьев, которые уменьшены наполовину и повернуты. Таким образом, имея достаточно мощную лупу, можно на странице бумаги изобразить бинарное дерево большого размера.

Это можно реализовать несколькими способами. Например, один из них заключается в представлении корня с помощью линии, проведенной от центра страницы в альбомной ориентации к ее верхнему краю. Причем представление левого поддерева получится в левой половине страницы после поворота на 90° по часовой стрелке, а правого поддерева — в правой половине страницы после поворота на 90° против часовой стрелки. Затем каждый узел будет представлен линией половинной длины (по сравнению с длиной предыдущего отрезка), проведенной снизу вверх. (Если применить этот метод для полного бинарного дерева (complete binary tree) с 2^k-1 узлами на k уровнях, получатся так называемые H-деревья (H-trees), которые представляют наиболее эффективную компоновку таких бинарных деревьев на СБИС-микросхеме (VLSI chip) (см. R. P. Brent and H. T. Kung, *Inf. Proc. Letters* **11** (1980), 46–48.)





Другой способ заключается в представлении пустого бинарного дерева в виде прямоугольника и в таком вращении представления для поддеревьев непустых бинарных деревьев, чтобы левые поддеревья располагались вперемешку слева или снизу от соответствующих правых поддеревьев в зависимости от четности очередного рекурсивного шага этого построения. В результате подобного построения прямоугольники будут соответствовать внешним узлам расширенного бинарного дерева (см. раздел 2.3.4.5). Это представление, в значительной степени связанное с понятиями 2-мерных (2-d trees) и четырехмерных (quadtrees) деревьев, которое обсуждается в разделе 6.5, особенно удобно использовать, когда информацию содержат внешние, а не внутренние узлы.

РАЗДЕЛ 2.3.1

1. `INFO(T)` $=A$, `INFO(RLINK(T))` $=C$ и т. д.; в результате получим H.

2. В прямом порядке обхода: 1245367; в симметричном порядке обхода: 4251637; в обратном порядке обхода: 4526731.

3. Да, справедливо. Обратите внимание, например в упр. 2, на то, что узлы 4–7 всегда располагаются в таком порядке. Этот результат можно получить автоматически методом индукции по размеру бинарного дерева.

4. Он является реверсивным по отношению к обратному порядку обхода. (Это легко можно доказать по индукции.)

5. Например, узлы дерева из упр. 2 в прямом порядке в двоичной системе (которая в данном случае эквивалентна десятичной системе обозначений Дьюи) выглядели бы так: 1, 10, 100, 101, 11, 110, 111. Здесь строки цифр рассортированы, как слова в словаре.

Вообще, при лексикографической сортировке слева направо узлы будут перечисляться в прямом порядке, если "пропуск" $< 0 < 1$, в обратном порядке, если $0 < 1 <$ "пропуск", и в симметричном порядке, если $0 <$ "пропуск" < 1.

(Более того, если представить пропуски слева и рассмотреть ярлыки в десятичной системе обозначений Дьюи как обычные числа в двоичной системе счисления, получим *порядок уровней* (*level order*); см. 2.3.3–(8).)

6. Тот факт, что последовательность $p_1 p_2 \dots p_n$ можно получить с помощью стека, легко доказывается методом индукции по n Действительно, нетрудно заметить, что именно эти действия выполняет стек в алгоритме T (Соответствующая последовательность символов S и X, как в упр. 2.2.1-3, также соответствует последовательности цифр 1 и 2, которые используются в качестве индексов в двойном порядке; см. упр. 18.)

И наоборот, если последовательность $p_2 \dots p_n$ можно получить с помощью стека и если $p_k = 1$, то $p_1 \dots p_{k-1}$ является перестановкой $\{2, \dots, k\}$ и $p_{k+1} \dots p_n$ — перестановкой $\{k+1, \dots, n\}$. Эти перестановки соответствуют левому и правому поддеревьям, и их можно получить с помощью стека. Доказательство можно выполнить по индукции.

7. Используя прямой порядок, находим корень, а затем на основе симметричного порядка — левое и правое поддеревья Таким образом получим прямой и симметричный порядки этих поддеревьев. Следовательно, структуру такого дерева можно легко восстановить (действительно, создание простого алгоритма восстановления структуры дерева по узлам, связанным в прямом порядке связями-нитями LLINK и в симметричном порядке связями-нитями RLINK, представляет собой довольно интересную задачу). Аналогично можно восстановить структуру дерева, зная симметричный и обратный порядки. Но знания прямого и обратного порядков недостаточно для восстановления структуры любого дерева, поскольку существует два бинарных дерева, которые выглядят, как AB, в прямом порядке и, как BA, в обратном порядке. Однако, если все неконцевые узлы бинарного дерева имеют *по две* непустые ветви, его структуру *можно* восстановить, зная прямой и обратный порядки

8. (a) Бинарные деревья со всеми пустыми связями LLINK. (b) Пустое бинарное дерево или дерево, содержащее только один узел (c) Бинарные деревья со всеми пустыми связями RLINK.

9. T1 выполняется один раз, T2 — $2n+1$ раз, T3 — n раз, T4 — $n+1$ раз, T5 — n раз. Эти значения можно получить либо методом индукции, либо с помощью закона Кирхгофа, либо непосредственно анализируя программу T.

10. При обходе бинарного дерева со всеми пустыми связями RLINK потребуется поместить в стек адреса всех n узлов до того, как какие-либо из них будут удалены.

11. Пусть a_{nk} — количество бинарных деревьев с n узлами, для которых стек в алгоритме T никогда не содержит более k узлов. Если $g_k(z) = \sum_n a_{nk} z^n$, то $g_1(z) = 1/(1-z)$, $g_2(z) = 1/(1 - z/(1-z)) = (1-z)/(1-2z)$, $\dots$, $g_k(z) = 1/(1 - z g_{k-1}(z)) = q_{k-1}(z)/q_k(z)$, где $q_{-1}(z) = q_0(z) = 1$, $q_{k+1}(z) = q_k(z) - z q_{k-1}(z)$. Следовательно,

$$g_k(z) = (f_1(z)^{k+1} - f_2(z)^{k+1})/(f_1(z)^{k+2} - f_2(z)^{k+2}),$$

где $f_j(z) = \frac{1}{2}(1 \pm \sqrt{1-4z}\,)$. Теперь можно показать, что

$$a_{nk} = [u^n](1-u)(1+u)^{2n}(1-u^{k+1})/(1-u^{k+2}).$$

Значит, $s_n = \sum_{k\ge 1} k(a_{nk} - a_{n(k-1)})$ равно $[u^{n+1}](1-u)^2(1+u)^{2n}\sum_{j\ge 1} u^j/(1-u^j)$ минус a_{nn}. Используя метод из упр. 5.2.2–52, получим асимптотический ряд

$$s_n/a_{nn} = \sqrt{\pi n} - \frac{3}{2} + \frac{11}{24}\sqrt{\frac{\pi}{n}} + O(n^{-3/2}).$$

[N. G. de Bruijn, D. E. Knuth, and S. O. Rice, in *Graph Theory and Computing*, ed. by R. C. Read (New York: Academic Press, 1972), 15–22.]

Если данное бинарное дерево представляет лес, описанный в разделе 2.3 2, то полученная величина является его *высотой* (*height*) (т. е. наибольшим расстоянием между корнем и узлами дерева плюс один). Обобщения этого результата для других вариантов деревьев были получены Флажоле и Одлыжко [*J. Computer and System Sci.* **25** (1982), 171–213]. Асимптотическое распределение высоты вблизи и вдали от среднего значения проанализированы Флажоле, Гао, Одлыжко и Ричмондом [*Combinatorics, Probability, and Computing* **2** (1993), 145–156].

12. Узел `NODE(P)` следует посещать между шагами T2 и T3, а не между шагами T4 и T2. Чтобы доказать справедливость этого предложения, докажите истинность утверждения "*Начиная с шага T2 с ... исходным состоянием стека* `A`, *содержащим элементы* `A[1]` ... `A[`m`]`" точно так, как в тексте данного раздела.

13. (Решение принадлежит C. Араухо (S. Araújo), 1976) Оставим шаги T1–T4 без изменений, но сделаем так, чтобы `Q` инициализировалось значением Λ на шаге T1 и `Q` указывало на последний посещенный узел, если такой имеется. Шаг T5 заменим двумя другими шагами.

T5. [Правая ветвь пройдена?] Если `RLINK(P)` $= \Lambda$ или `RLINK(P)` $=$ `Q`, перейти к шагу T6; в противном случае установить `A` $\Leftarrow$ `P`, `P` $\leftarrow$ `RLINK(P)` и вернуться к шагу T2.

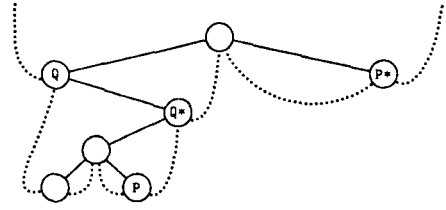
T6. [Попасть в P.] Попасть в узел `NODE(P)`, установить `Q` $\leftarrow$ `P` и вернуться к шагу T4

Здесь применяется аналогичное доказательство. (Шаги T4 и T5 можно упростить таким образом, чтобы исключить операции извлечения узлов из стека с их немедленной повторной вставкой в стек.)

14. По методу индукции всегда существует в точности $n+1$ Λ-связей (учитывая `T`, когда эта связь пуста). Так как с учетом `T` существует n непустых связей, замечание в данном разделе о преобладании пустых связей является справедливым.

15. Связь-нить `LLINK` или `RLINK` указывает на узел тогда и только тогда, когда он имеет непустое правое или левое поддерево соответственно (см. рис. 24).

16. Если `LTAG(Q)` $= 0$, то `Q*=LLINK(Q)`. Таким образом, узел `Q*` расположен на один шаг ниже и левее. В противном случае `Q*` можно найти, если пройти вверх по дереву (если это необходимо) повторно до первой возможности перейти вниз и вправо без выполнения повторных шагов. Типичными примерами таких проходов являются пути от `P` к `P*` и от `Q` к `Q*` в следующем дереве



17. Если `LTAG(P)` $= 0$, установить `Q` $\leftarrow$ `LLINK(P)` и прекратить выполнение алгоритма. В противном случае установить `Q` $\leftarrow$ `P`, затем ни разу или несколько раз устанавливать

Q ← RLINK(Q) до тех пор, пока не будет соблюдено условие RTAG(Q) = 0. Наконец, еще один раз установить Q ← RLINK(Q).

18. Измените алгоритм T, вставив следующий шаг T2.5: "Попасть в узел NODE(P) (в первый раз)", а на шаге T5 узел NODE(P) будет посещен уже во второй раз.

Обход прошитого дерева выполняется очень просто:

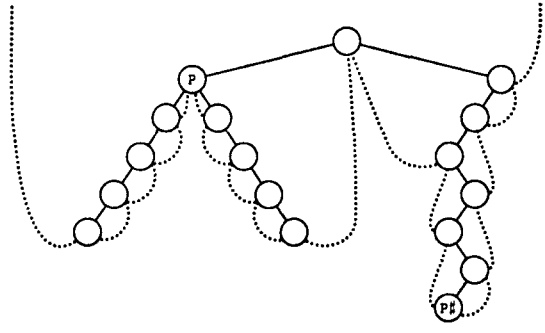
$$(P, 1)^{\triangle} = (\texttt{LLINK(P)}, 1), \text{ если } \texttt{LTAG(P)} = 0, \text{ в противном случае } (P, 2);$$

$$(P, 2)^{\triangle} = (\texttt{RLINK(P)}, 1), \text{ если } \texttt{RTAG(P)} = 0, \text{ в противном случае } (\texttt{RLINK(P)}, 2).$$

В любом случае в дереве происходит переход только на один шаг, поэтому на самом деле двойной порядок и значения d и e включаются в программу без явного упоминания.

Опуская все посещения узлов в первый раз, получим в точности алгоритмы T и S, а опуская все посещения узлов во второй раз, получим решения упр. 12 и 17.

19. Основная идея заключается в поиске родителя Q узла P. Если $P \neq$ LLINK(Q), то получим P♯ = Q; в противном случае получим P♯, повторно устанавливая Q ← Q$ несколько раз (может быть, это вовсе не потребуется делать) до тех пор, пока не выполнится условие RTAG(Q) = 1. (См., например, P и P♯ в показанном ниже дереве.)



Для общего случая правопрошитого дерева не существует эффективного алгоритма поиска родителя узла P, так как вырожденное правопрошитое дерево, в котором все левые связи пусты, является циклическим списком, в котором связи проведены в другом направлении. Следовательно, нельзя совершить обход правопрошитого дерева в обратном порядке с такой же эффективностью, как при использовании стека в упр. 13, без сохранения информации о пути к текущему узлу P.

Но если дерево прошито в обоих направлениях, родителя узла P *можно* найти следующим достаточно эффективным способом.

F1. Установить Q ← P и R ← P.

F2. Если LTAG(Q) = RTAG(R) = 0, установить Q ← LLINK(Q) и R ← RLINK(R) и повторить этот шаг. В противном случае перейти к шагу F4, если RTAG(R) = 1.

F3. Установить Q ← LLINK(Q) и прекратить выполнение алгоритма, если P = RLINK(Q). В противном случае не устанавливать или устанавливать R ← RLINK(R) до тех пор, пока не выполнится условие RTAG(R) = 1; затем установить Q ← RLINK(R) и прекратить выполнение алгоритма.

F4. Установить R ← RLINK(R) и прекратить выполнение алгоритма с Q ← R, если P = LLINK(R) В противном случае не устанавливать или устанавливать Q ← LLINK(Q) до тех пор, пока не выполнится условие LTAG(Q) = 1; затем установить Q ← LLINK(Q) и прекратить выполнение алгоритма. ▮

Среднее время выполнения алгоритма F для произвольного узла P дерева равно $O(1)$. Действительно, если подсчитать только шаги Q ← LLINK(Q), когда P — правый потомок,

или только шаги R ← RLINK(R) когда P — левый потомок, то обход каждой связи будет выполнен в точности для одного узла P.

20. Строки 06–09 нужно заменить строками

```
T3  ENT4  0,6
    LD6   AVAIL
    J6Z   OVERFLOW
    LDX   0,6(LINK)
    STX   AVAIL
    ST5   0,6(INFO)
    ST4   0,6(LINK)
```

Строки 12 и 13 нужно заменить строками

```
    LD4   0,6(LINK)
    LD5   0,6(INFO)
    LDX   AVAIL
    STX   0,6(LINK)
    ST6   AVAIL
    ENT6  0,4
```

Если в строку 06 добавить еще две строки

```
T3  LD3   0,5(LLINK)
    J3Z   T5              Перейти к шагу T5, если LLINK(P) = Λ.
```

с соответствующими изменениями строк 10 и 11, то время выполнения программы $(30n + a + 4)u$ сократится до $(27a + 6n - 22)u$. (Аналогично можно было сократить время выполнения программы T до $(12a + 6n - 7)u$, что несколько меньше при $a = (n + 1)/2$.)

21. Приведенное ниже решение Джозефа М. Морриса [Joseph M. Morris, *Inf. Proc. Letters* **9** (1979), 197–200] позволяет совершить обход только в прямом порядке (см. упр. 18).

U1. [Инициализация.] Установить P ← T и R ← Λ.

U2. [Обход завершен?] Если P = Λ, прекратить выполнение алгоритма.

U3. [Обход слева.] Установить Q ← LLINK(P). Если Q = Λ, посетить узел NODE(P) в прямом порядке и перейти к шагу U6.

U4. [Поиск связи-нити.] Ни разу не устанавливать или устанавливать Q ← RLINK(Q) до тех пор, пока не выполнится либо условие Q = R, либо условие RLINK(Q) = Λ.

U5. [Вставка или удаление связи-нити.] Если Q ≠ R, установить RLINK(Q) ← P и перейти к шагу U8. В противном случае установить RLINK(Q) ← Λ (связь-нить временно была установлена в P, а теперь совершим обход левого поддерева P).

U6. [Посещение в симметричном порядке.] Посетить NODE(P) в симметричном порядке.

U7. [Переход вверх или вправо.] Установить R ← P, P ← RLINK(P) и вернуться к шагу U2.

U8. [Посещение в прямом порядке.] Посетить NODE(P) в прямом порядке.

U9. [Переход влево.] Установить P ← LLINK(P) и вернуться к шагу U3. ▮

Моррис также предложил несколько более сложный способ обхода дерева в обратном порядке.

Совершенно иное решение было найдено Дж. М. Робсоном [J. M. Robson, *Inf. Proc. Letters* **2** (1973), 12–14]. Назовем узел заполненным, если его связи LLINK и RLINK не пусты, и назовем его пустым, если обе его связи LLINK и RLINK пусты. Робсон нашел способ организации стека указателей на заполненные узлы, правые поддеревья которых посещаются во время обхода дерева, используя поля связи из пустых узлов!

Еще один способ избежать использования вспомогательного стека был независимо найден Г. Э. Линдстромом и Б. Двайером [G. Lindstrom and B. Dwyer, *Inf. Proc. Letters* **2** (1973), 47–51, 143–145]. В их алгоритме обход дерева совершается в *тройном порядке* (*triple order*), т. е. каждый узел посещается в точности три раза: один раз — в прямом,

второй — в симметричном, а третий — в обратном порядке. Но при этом неизвестно, какой из порядков задействован для посещения узла в текущий момент.

W1. [Инициализация.] Установить P ← T и Q ← S, где S — это значение метки, т. е. любое число, которое наверняка отличается от значения любой другой связи в данном дереве (например, -1).

W2. [Пропуск пустой связи.] Если P = Λ, установить P ← Q и Q ← Λ.

W3. [Обход завершен?] Если P = S, прекратить выполнение алгоритма. (По завершении работы алгоритма получим Q = T.)

W4. [Посещение узла.] Попасть в узел NODE(P).

W5. [Обращение.] Установить R ← LLINK(P), LLINK(P) ← RLINK(P), RLINK(P) ← Q, Q ← P, P ← R и вернуться к шагу W2. ▮

Справедливость алгоритма следует из того, что если начать шаг W2 со значением P, указывающим на корень бинарного дерева T, и значением Q, указывающим на X, где X не является связью в этом дереве, то данный алгоритм трижды выполнит обход дерева и достигнет шага W3 со значениями P = X и Q = T.

Если $\alpha(\mathtt{T}) = x_1 x_2 \ldots x_{3n}$ является результирующей последовательностью узлов в тройном порядке обхода, то в таком случае получим $\alpha(\mathtt{T}) = \mathtt{T}\,\alpha(\mathtt{LLINK(T)})\,\mathtt{T}\,\alpha(\mathtt{RLINK(T)})\,\mathtt{T}$. Следовательно, как и заметил Линдстром, три подпоследовательности $x_1 x_4 \ldots x_{3n-2}$, $x_2 x_5 \ldots x_{3n-1}$, $x_3 x_6 \ldots x_{3n}$ содержат все узлы дерева, причем каждый из них встречается в них всего один раз. (Так как узел x_{j+1} является либо родителем, либо потомком узла x_j, эти подпоследовательности посещают узлы таким образом, что каждый из них находится на расстоянии не более трех связей от своего предшественника. В разделе 7.2.1 описывается общая схема обхода, которая называется *прямообратным порядком* (*prepostorder*) и обладает этим свойством не только в отношении бинарных деревьев, но и деревьев общего типа.)

22. В этой программе применяются обозначения и соглашения, используемые в программах T и S, с Q в регистре rI6 и/или rI4. Старомодный компьютер MIX не очень подходит для сравнения индексных регистров, поэтому переменная R опущена и проверка условия Q = R заменена проверкой условия RLINK(Q) = P.

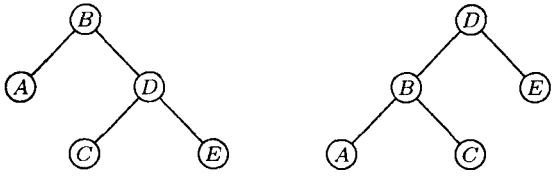
```
01  U1   LD5  HEAD(LLINK)      1            U1. Инициализация. P ← T.
02  U2A  J5Z  DONE             1            Если P = Λ, прекратить
                                               выполнение алгоритма.
03  U3   LD6  0,5(LLINK)       n+a-1        U3. Обход слева. Q ← LLINK(P).
04       J6Z  U6                n+a-1        Перейти к шагу U6, если Q = Λ.
05  U4   CMP5 1,6(RLINK)       2n-2b        U4. Поиск связи-нити.
06       JE   5F                2n-2b        Совершить переход, если RLINK(Q) = P.
07       ENT4 0,6               2n-2b-a+1    rI4 ← Q.
08       LD6  1,6(RLINK)        2n-2b-a+1
09       J6NZ U4                2n-2b-a+1    Перейти к шагу U4 с Q ← RLINK(Q),
                                               если это значение ≠ 0.
10  U5   ST5  1,4(RLINK)       a-1          U5a. Вставка связи-нити. RLINK(Q) ← P.
11  U9   LD5  0,5(LLINK)       a-1          U9. Переход влево. P ← LLINK(P).
12       JMP  U3                a-1          Перейти к шагу U3.
13  5H   STZ  1,6(RLINK)       a-1          U5b. Удаление связи-нити. RLINK(Q) ← Λ.
14  U6   JMP  VISIT            n            U6. Посещение в симметричном порядке.
15  U7   LD5  1,5(RLINK)       n            U7. Переход вверх или вправо. P ← RLINK(P)
16  U2   J5NZ U3               n            U2. Обход завершен? Перейти к шагу U3,
                                               если P ≠ Λ.
17  DONE ...                                 ▮
```

Общее время выполнения равно $21n+6a-3-14b$, где n — количество узлов, a — количество пустых связей RLINK (следовательно, $a-1$ — это количество непустых связей LLINK), b — количество узлов на "правом хребте" дерева T, RLINK(T), RLINK(RLINK(T)) и т. д.

23. Вставка узла справа: RLINKT(Q) $\leftarrow$ RLINKT(P), RLINK(P) $\leftarrow$ Q, RTAG(P) $\leftarrow$ 0, LLINK(Q) $\leftarrow \Lambda$. При вставке узла слева при условии, что LLINK(P) $= \Lambda$, нужно выполнить следующие действия: установить LLINK(P) $\leftarrow$ Q, LLINK(Q) $\leftarrow \Lambda$, RLINK(Q) $\leftarrow$ P, RTAG(Q) $\leftarrow 1$. Вставка узла слева между P и LLINK(P) $\neq \Lambda$: установить R $\leftarrow$ LLINK(P), LLINK(Q) $\leftarrow$ R, а затем *не* устанавливать или устанавливать R $\leftarrow$ RLINK(R) до тех пор, пока не выполнится условие RTAG(R) $= 1$; и наконец, установить RLINK(R) $\leftarrow$ Q, LLINK(P) $\leftarrow$ Q, RLINK(Q) $\leftarrow$ P, RTAG(Q) $\leftarrow 1$.

(В последнем случае можно использовать более эффективный алгоритм, если известен такой узел F, что P = LLINK(F) или P = RLINK(F). Например, если предположить, что выполняется последнее из этих двух равенств, можно установить INFO(P) $\leftrightarrow$ INFO(Q), RLINK(F) $\leftarrow$ Q, LLINK(Q) $\leftarrow$ P, RLINK(P) $\leftarrow$ Q, RTAG(P) $\leftarrow 1$. Хотя для выполнения этих действий потребуется фиксированное время, в общем случае их не рекомендуется использовать, поскольку при этом повсюду в памяти компьютера придется выполнять интенсивный обмен содержимым узлов.)

24. Нет, как видно из этого примера.



25. Сначала методом индукции по количеству узлов дерева T докажем (b), а затем аналогично — (c). Для доказательства (a) рассмотрим несколько случаев Обозначим $T \preceq_1 T'$, если выполняется условие (i), обозначим $T \preceq_2 T'$, если выполняется условие (ii), и т. д. Тогда из $T \preceq_1 T'$ и $T' \preceq T''$ следует $T \preceq_1 T''$; $T \preceq_2 T'$ и из $T' \preceq T''$ следует $T \preceq_2 T''$; оставшиеся два случая доказываются методом индукции по количеству узлов в дереве T.

26. *Если в результате двойного порядка обхода дерева T получена последовательность* $(u_1, d_1), (u_2, d_2), \ldots, (u_{2n}, d_{2n})$, где u — это узлы и d_k равны 1 или 2, построим "след" дерева $(v_1, s_1), (v_2, s_2), \ldots, (v_{2n}, s_{2n})$, где $v_j = \text{info}(u_j)$, а $s_j = l(u_j)$ или $r(u_j)$ *в зависимости от того,* $d_j = 1$ или $d_j = 2$ соответственно. Теперь $T \preceq T'$ тогда и только тогда, когда определенный выше след дерева T *лексикографически* предшествует или равен следу дерева T'. Формально это значит, что либо $n \le n'$ и $(v_j, s_j) = (v'_j, s'_j)$ для $1 \le j \le n$, либо есть такое k, для которого $(v_j, s_j) = (v'_j, s'_j)$ для $1 \le j < k$ и либо $v_k \prec v'_k$, либо $v_k = v'_k$ и $s_k < s'_k$.

27. **R1.** [Инициализация] Установить P $\leftarrow$ HEAD, P′ $\leftarrow$ HEAD′; это заголовки соответствующих списков данных правопрошитых бинарных деревьев. Перейти к шагу R3.

R2. [Проверка поля INFO.] Если INFO(P) $\prec$ INFO(P′), прекратить выполнение алгоритма ($T \prec T'$); если INFO(P) $\succ$ INFO(P′), прекратить выполнение алгоритма ($T \succ T'$).

R3. [Переход влево] Если LLINK(P) $= \Lambda =$ LLINK(P′), перейти к шагу R4; если LLINK(P) $= \Lambda \neq$ LLINK(P′), прекратить выполнение алгоритма ($T \prec T'$), если LLINK(P) $\neq \Lambda =$ LLINK(P′), прекратить выполнение алгоритма ($T \succ T'$); в противном случае установить P $\leftarrow$ LLINK(P), P′ $\leftarrow$ LLINK(P′) и перейти к шагу R2.

R4. [Конец обхода?] Если P = HEAD (или, что эквивалентно, если P′ = HEAD′), прекратить выполнение алгоритма (T эквивалентно T').

R5. [Переход вправо.] Если RTAG(P) = 1 = RTAG(P′), установить P ← RLINK(P), P′ ← RLINK(P′) и перейти к шагу R4. Если RTAG(P) = 1 ≠ RTAG(P′), прекратить выполнение алгоритма ($T \prec T'$). Если RTAG(P) ≠ 1 = RTAG(P′), прекратить выполнение алгоритма ($T \succ T'$). В противном случае установить P ← RLINK(P), P′ ← RLINK(P′) и перейти к шагу R2. ▮

Для доказательства справедливости этого алгоритма (и, следовательно, для понимания принципа его работы) можно методом индукции по размеру дерева T_0 показать, что справедливо следующее утверждение: *"Начиная с шага R2 с* P *и* P′*, указывающими на корни двух непустых правопрошитых бинарных деревьев T_0 и T_0', выполнение алгоритма прекращается, если T_0 и T_0' не эквивалентны, но известно, что либо $T_0 \prec T_0'$, либо $T_0 \succ T_0'$. Алгоритм достигнет шага R4, если T_0 и T_0' являются эквивалентными с* P *и* P′*, указывающими соответственно на узлы-последователи для узлов T_0 и T_0' в симметричном порядке"*.

28. Исходное дерево эквивалентно u подобно его копии.

29. Докажите методом индукции по размеру дерева T, что справедливо следующее утверждение: *"Начиная с шага C2 с* P*, указывающим на корень непустого бинарного дерева T, с* Q*, указывающим на узел, который имеет пустые левое и правое поддеревья, эта процедура в конечном итоге достигнет шага C6 после установки* INFO(Q) ← INFO(P) *и присоединения копий левого и правого поддеревьев узла* NODE(P) *к узлу* NODE(Q) *и с* P *и* Q*, указывающими соответственно на узлы-предшественники деревьев T и узла* NODE(Q) *в прямом порядке"*.

30. Предположим, что указатель T в (2) равен LLINK(HEAD) в (10).

L1. [Инициализация.] Установить Q ← HEAD, RLINK(Q) ← Q.

L2. [Продвижение.] Установить P ← Q$ (см. ниже).

L3. [Прошивка.] Если RLINK(Q) = Λ, установить RLINK(Q) ← P, RTAG(Q) ← 1; в противном случае установить RTAG(Q) ← 0. Если LLINK(P) = Λ, установить LLINK(P) ← Q, LTAG(P) ← 1; в противном случае установить LTAG(P) ← 0.

L4. [Обход завершен?] Если P ≠ HEAD, установить Q ← P и вернуться к шагу L2. ▮

На шаге L2 этого алгоритма предполагается активизация сопрограммы симметричного порядка обхода, как в алгоритме T, с дополнительным условием, что алгоритм T посещает узел HEAD после полного обхода всего дерева. Это обозначение является удобным упрощением при описании алгоритмов обхода деревьев, поскольку не требуется повторять описание стека из алгоритма T. Конечно, на шаге L2 не может использоваться алгоритм S, поскольку дерево еще не прошито. Но алгоритм из упр. 21 использовать *можно*, и благодаря этому появляется еще один удобный метод прошивки дерева без какого-либо вспомогательного стека.

31. **X1.** Установить P ← HEAD.

X2. Установить Q ← P$ (используя, например, алгоритм S модифицированного правопрошитого дерева).

X3. Если P ≠ HEAD, установить AVAIL ⇐ P.

X4. Если Q ≠ HEAD, установить P ← Q и вернуться к шагу X2.

X5. Установить LLINK(HEAD) ← Λ. ▮

Очевидно, что возможны и другие решения, которые сокращают протяженность внутреннего цикла, хотя порядок основных шагов довольно сомнителен. Указанная процедура работоспособна, поскольку ни один узел не возвращается в область доступных ячеек до тех пор, пока алгоритм S не исследует его связи LLINK и RLINK. Как упоминалось в тексте данного раздела, каждая из этих связей используется в точности один раз при полном обходе дерева.

32. RLINK(Q) $\leftarrow$ RLINK(P), SUC(Q) $\leftarrow$ SUC(P), SUC(P) $\leftarrow$ RLINK(P) $\leftarrow$ Q, PRED(Q) $\leftarrow$ P, PRED(SUC(Q)) $\leftarrow$ Q.

33. Вставка узла NODE(Q) слева и снизу узла NODE(P) выполняется довольно просто: установить LLINKT(Q) $\leftarrow$ LLINKT(P), LLINK(P) $\leftarrow$ Q, LTAG(P) $\leftarrow$ 0, RLINK(Q) $\leftarrow \Lambda$. Вставку справа выполнить гораздо сложнее, поскольку для этого необходимо найти *Q, что так же сложно выполнить, как и найти Q♯ (см. упр. 19). Для этого можно использовать метод перемещения узлов, который рассматривается в упр. 23. Поэтому в общем случае вставки при таком типе прошивки выполняются гораздо сложнее. Но вставки согласно алгоритму C выполняются гораздо проще, чем вставки общего типа. Действительно, процесс копирования осуществляется несколько быстрее для такого типа прошивки.

C1. Установить P $\leftarrow$ HEAD, Q $\leftarrow$ U, перейти к шагу C4. (Далее используются предположения и идеология алгоритма C из данного раздела.)

C2. Если RLINK(P) $\neq \Lambda$, установить R $\Leftarrow$ AVAIL, LLINK(R) $\leftarrow$ LLINK(Q), LTAG(R) $\leftarrow$ 1, RLINK(R) $\leftarrow \Lambda$, RLINK(Q) $\leftarrow$ LLINK(Q) $\leftarrow$ R.

C3. Установить INFO(Q) $\leftarrow$ INFO(P).

C4. Если LTAG(P) = 0, установить R $\Leftarrow$ AVAIL, LLINK(R) $\leftarrow$ LLINK(Q), LTAG(R) $\leftarrow$ 1, RLINK(R) $\leftarrow \Lambda$, LLINK(Q) $\leftarrow$ R, LTAG(Q) $\leftarrow$ 0.

C5. Установить P $\leftarrow$ LLINK(P), Q $\leftarrow$ LLINK(Q).

C6. Если P $\neq$ HEAD, перейти к шагу C2. ▮

Теперь этот алгоритм выглядит настолько простым, что возникают сомнения в его корректности!

Алгоритм C для прошитых или правопрошитых бинарных деревьев выполняется несколько дольше из-за дополнительных затрат времени на вычисление P*, Q* на шаге C5.

Можно обычным образом прошить связи RLINK или поместить ♯P в RLINK(P) в сочетании с методом копирования за счет заданных соответствующим образом значений RLINK(R) и RLINKT(Q) на шагах C2 и C4.

34. A1. Установить Q $\leftarrow$ P, а затем ни разу не устанавливать или устанавливать Q $\leftarrow$ RLINK(Q) до тех пор, пока не выполнится условие RTAG(Q) = 1.

A2. Установить R $\leftarrow$ RLINK(Q). Если LLINK(R) = P, установить LLINK(R) $\leftarrow \Lambda$. В противном случае установить R $\leftarrow$ LLINK(R), затем ни разу не устанавливать или устанавливать R $\leftarrow$ RLINK(R) до тех пор, пока не выполнится условие RLINK(R) = P; наконец, установить RLINKT(R) $\leftarrow$ RLINKT(Q). (На этом шаге узел NODE(P) и его поддеревья удаляются из исходного дерева.)

A3. Установить RLINK(Q) $\leftarrow$ HEAD, LLINK(HEAD) $\leftarrow$ P. ▮

(Ключом к успеху при разработке и/или понимании этого алгоритма является построение достаточно наглядных схем состояний "до и после".)

36. Нет; см. ответ к упр. 1.2.1–15(e).

37. Если LLINK(P) = RLINK(P) = Λ для представления (2), положим LINK(P) = Λ; в противном случае положим *LINK*(P) = Q, где *NODE*(Q) соответствует *NODE*(*LLINK*(P)) и *NODE*(Q+1) соответствует NODE(RLINK(P)). Условие LLINK(P) или RLINK(P) = Λ представлено с помощью признака в узле NODE(Q) или NODE(Q+1) соответственно. Для этого представления потребуется от n до $2n-1$ ячеек памяти. Для принятых допущений в случае (2) потребовалось бы 18 слов памяти по сравнению с 11 словами в данной схеме. При этом операции вставки и удаления будут выполняться приблизительно с одинаковой эффективностью в любом представлении. Но данное представление не так универсально при совместной работе с другими структурами.

РАЗДЕЛ 2.3.2

1. Если B — пустое дерево, то $F(B)$ — пустой лес. В противном случае лес $F(B)$ состоит из дерева T и леса $F(\text{правое поддерево}(B))$, где $\text{корень}(T) = \text{корень}(B)$ и $\text{поддеревья}(T) = F(\text{поддерево}(B))$.

2. Количество нулей в двоичном формате представления деревьев соответствует количеству десятичных разделительных знаков в десятичной системе обозначений. Точная формула такого соответствия выглядит так:

$$a_1.a_2.\cdots.a_k \leftrightarrow 1^{a_1}01^{a_2-1}0\ldots01^{a_k-1},$$

где 1^a обозначает a расположенных подряд единиц.

3. Рассортируем в лексикографическом порядке обозначения узлов в десятичной системе обозначений Дьюи (слева направо, как в словаре), размещая более короткую последовательность $a_1.\cdots.a_k$ перед ее расширениями $a_1.\cdots.a_k.\cdots.a_r$ для прямого порядка обхода и после расширений — для обратного порядка обхода узлов. Таким образом, при сортировке слов, а не последовательностей чисел, для прямого порядка обхода потребовалось бы разместить слова *кат* и *катаракта* в обычном порядке следования слов в словаре. А для получения обратного порядка обхода придется обратить исходное расположение слов: *катаракта*, *кат*. Эти правила легко доказать методом индукции по размеру дерева.

4. Да, справедливо, и это легко доказать методом индукции по количеству узлов дерева.

5. (a) Симметричный. (b) Обратный. Формулировка строгого доказательства эквивалентности этих алгоритмов обхода на основе метода индукции представляет собой довольно интересную задачу.

6. Прямой порядок обхода дерева T = прямой порядок обхода дерева T', обратный порядок обхода дерева T = симметричный порядок обхода дерева T', даже если T имеет узлы только с одним ребенком. Остальные два порядка не имеют простой взаимосвязи, например корень дерева T находится в конце в одном случае и приблизительно в середине — в другом случае.

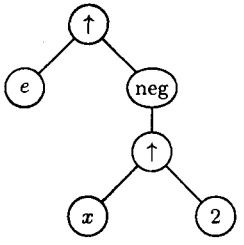
7. (a) Да, (b) нет, (c) нет, (d) да. Обратите внимание, что порядок, реверсивный прямому порядку обхода леса, эквивалентен обратному порядку правопрошитого реверсивного леса (как при зеркальном отражении).

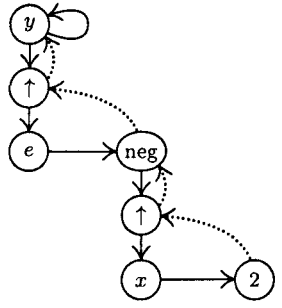
8. $T \preceq T'$ означает, что либо $\text{info}(\text{корень}(T)) \prec \text{info}(\text{корень}(T'))$, либо информация, содержащаяся в корнях, эквивалентна и выполняется следующее условие: предположим, что $T_1, \ldots, T_n$ являются поддеревьями корня дерева T, а $T'_1, \ldots, T'_{n'}$ — поддеревьями корня дерева T', и допустим, что $k \geq 0$ такое максимально возможное, что T_j эквивалентно T'_j для $1 \leq j \leq k$. Тогда либо $k = n$, либо $k < n$ и $T_{k+1} \preceq T'_{k+1}$.

9. В непустом лесу количество неконцевых узлов на один меньше количества правых связей, равных Λ, потому что все пустые правые связи соответствуют крайнему справа ребенку каждого неконцевого узла, а также корню крайнего справа дерева в этом лесу. (На основании данного факта можно привести еще одно доказательство упр. 2.3.1–14, так как количество пустых левых связей, очевидно, равно количеству *концевых* узлов.)

10. Эти леса подобны тогда и только тогда, когда $n = n'$ и $d(u_j) = d(u'_j)$ для $1 \leq j \leq n$; они эквивалентны тогда и только тогда, когда, кроме того, $\text{info}(u_j) = \text{info}(u'_j)$, $1 \leq j \leq n$. Это доказательство выполняется аналогично предыдущему доказательству на основе обобщения леммы 2.3.1P, где следует допустить, что $f(u) = d(u) - 1$.

11.





12. Если INFO(Q1) $\neq 0$, установить R $\leftarrow$ COPY(P1); если TYPE(P2) $= 0$ и INFO(P2) $\neq 2$, установить R $\leftarrow$ TREE("↑",TREE(INFO(P2) $-$ 1)); если TYPE(P2) $\neq 0$, установить R $\leftarrow$ TREE("↑",R, TREE("$-$",COPY(P2),TREE(1))); затем установить Q1 $\leftarrow$ MULT(Q1,MULT(COPY(P2),R)).

Если INFO(Q) $\neq 0$, установить Q $\leftarrow$ TREE("$\times$",MULT(TREE("ln",COPY(P1)),Q),TREE("↑", COPY(P1),COPY(P2))).

Наконец, перейти к шагу DIFF[4].

13. Приведенная ниже программа реализует алгоритм 2.3.1C с регистрами rI1 $\equiv$ P, rI2 $\equiv$ Q, rI3 $\equiv$ R и соответствующим образом изменяет условия инициализации и прекращения выполнения.

```
064       ST3  6F(0:2)        Сохранить содержимое регистров rI3, rI2.
065       ST2  7F(0:2)        C1. Инициализация.
066       ENT2 8F             Начать с создания узла NODE(U) с
067       JMP  1F               RLINK(U) = Λ.
068   8H  CON  0               Нулевая константа инициализации.
069   4H  LD1  0,1(LLINK)     Установить P ← LLINK(P) = P*.
070   1H  LD3  AVAIL          R ⇐ AVAIL.
071       J3Z  OVERFLOW
072       LDA  0,3(LLINK)
073       STA  AVAIL
074       ST3  0,2(LLINK)     LLINK(Q) ← R.
075       ENNA 0,2
076       STA  0,3(RLINKT)    RLINK(R) ← Q, RTAG(R) ← 1.
077       INCA 8B             rA ← LOC(исходный узел) − Q.
078       ENT2 0,3            Установить Q ← R = Q*.
079       JAZ  C3             Перейти к шагу C3 впервые.
080   C2  LDA  0,1            C2. Есть ли узел справа?
081       JAN  C3             Если RTAG(P) = 1, выполнить переход.
082       LD3  AVAIL          R ⇐ AVAIL.
083       J3Z  OVERFLOW
084       LDA  0,3(LLINK)
085       STA  AVAIL
086       LDA  0,2(RLINKT)
087       STA  0,3(RLINKT)    Установить RLINKT(R) ← RLINKT(Q).
088       ST3  0,2(RLINKT)    RLINK(Q) ← R, RTAG(Q) ← 0.
089   C3  LDA  1,1            C3. Копирование данных, т. е. поля INFO.
090       STA  1,2            Поле INFO скопировано.
091       LDA  0,1(TYPE)
092       STA  0,2(TYPE)      Поле TYPE скопировано.
093   C4  LDA  0,1(LLINK)     C4. Есть ли узел слева?
094       JANZ 4B             Если LLINK(P) ≠ Λ, выполнить переход.
```

```
095      STZ  0,2(LLINK)   LLINK(Q) ← Λ.
096  C5  LD2N 0,2(RLINKT)  C5. Продвинуться. Q ← −RLINKT(Q).
097      LD1  0,1(RLINK)   P ← RLINK(P).
098      J2P  C5           Если RTAG(Q) равно 1, то совершить переход.
099      ENN2 0,2          Q ← −Q.
100  C6  J2NZ C2           C6. Проверка завершения.
101      LD1  8B(LLINK)    rI1 ← адрес первого созданного узла.
102  6H  ENT3 *            Восстановить индексные регистры.
103  7H  ENT2 *            ▮
```

14. Пусть a — количество скопированных неконцевых узлов (т. е. узлов с операторами). Попыток выполнения разных строк из предыдущей программы будет столько: 064–067, 1; 069, a; 070–079, $a+1$; 080–081, $n-1$; 082–088, $n-1-a$; 089–094, n; 095, $n-a$; 096–098, $n+1$; 099–100, $n-a$; 101–103, 1. Общее время равняется $(36n+22)u$, при этом около 20% времени уходит на поиск свободных узлов, около 40% — на обход и около 40% — на копирование данных из полей INFO и LINK.

15. Читателю предлагается в качестве упражнения самостоятельно изучить этот код.

```
218  DIV LDA  1,6          232  1H ENTX *
219      JAZ  1F           233      JMP  TREE2
220      JMP  COPYP2       234      ST1  1F(0:2)
221      ENTA SLASH        235      JMP  COPYP1
222      ENTX 0,6          236      ENTA 0,1
223      JMP  TREE2        237      ENT1 0,5
224      ENT6 0,1          238      JMP  MULT
225  1H  LDA  1,5          239      ENTX 0,1
226      JAZ  SUB          240  1H  ENT1 *
227      JMP  COPYP2       241      ENTA SLASH
228      ST1  1F(0:2)      242      JMP  TREE2
229      ENTA CON2         243      ENT5 0,1
230      JMP  TREE0        244      JMP  SUB  ▮
231      ENTA UPARROW
```

16. Читателю предлагается в качестве упражнения самостоятельно изучить этот код.

```
245  PWR  LDA  1,6          263      JMP  TREE0        281      ENTA LOG
246       JAZ  4F           264  1H  ENTX *           282      JMP  TREE1
247       JMP  COPYP1       265      ENTA MINUS        283      ENTA 0,1
248       ST1  R(0:2)       266      JMP  TREE2        284      ENT1 0,5
249       LDA  0,3(TYPE)    267  5H  LDX  R(0:2)      285      JMP  MULT
250       JANZ 2F           268      ENTA UPARROW      286      ST1  1F(0:2)
251       LDA  1,3          269      JMP  TREE2        287      JMP  COPYP1
252       DECA 2            270      ST1  R(0:2)       288      ST1  2F(0:2)
253       JAZ  3F           271  3H  JMP  COPYP2       289      JMP  COPYP2
254       INCA 1            272      ENTA 0,1          290  2H  ENTX *
255       STA  CON0+1       273  R   ENT1 *           291      ENTA UPARROW
256       ENTA CON0         274      JMP  MULT         292      JMP  TREE2
257       JMP  TREE0        275      ENTA 0,6          293  1H  ENTX *
258       STZ  CON0+1       276      JMP  MULT         294      ENTA TIMES
259       JMP  5F           277      ENT6 0,1          295      JMP  TREE2
260  2H   JMP  COPYP2       278  4H  LDA  1,5          296      ENT5 0,1
261       ST1  1F(0:2)      279      JAZ  ADD          297      JMP  ADD  ▮
262       ENTA CON1         280      JMP  COPYP1
```

17. Ссылки на более ранние работы по этой теме можно найти в обзоре J. Sammet, *CACM* **9** (1966), 555–569.

18. Сначала следует таким образом установить LLINK[j] $\leftarrow$ RLINK[j] $\leftarrow$ j для всех j, чтобы каждый узел находился в циклическом списке длиной 1. Тогда для $j = n, n-1, \ldots, 1$ (в таком порядке), если PARENT[j] $= 0$, установить $r \leftarrow j$, в противном случае вставить циклический список, начиная с j, в циклический список, начиная с PARENT[j], в следующем порядке: $k \leftarrow$ PARENT[j], $l \leftarrow$ RLINK[k], $i \leftarrow$ LLINK[j], LLINK[j] $\leftarrow k$, RLINK[k] $\leftarrow j$, LLINK[l] $\leftarrow i$, RLINK[i] $\leftarrow l$. Этот алгоритм корректен, так как (a) каждый некорневой узел всегда располагается перед своим родителем или последователем своего родителя; (b) узлы каждого семейства располагаются в списке родителя в порядке их следования; (c) прямой порядок является единственным порядком, который удовлетворяет условиям (a) и (b).

20. *Если u является предком узла v, то по индукции автоматически следует, что узел u* предшествует узлу v в прямом порядке обхода и следует за узлом v в обратном порядке. И наоборот, если узел u предшествует узлу v в прямом порядке и следует за узлом v в обратном порядке, то докажем, что узел u является предком узла v. Это утверждение очевидно, если узел u является корнем первого дерева. Если u не является корнем первого дерева, то и узел v не будет корневым, так как u следует за узлом v в обратном порядке; далее доказательство выполняется по индукции. Аналогично, если u не принадлежит первому дереву, то и узел v не принадлежит ему, так как узел u предшествует узлу v в прямом порядке. (Результат этого упражнения также следует из результата упр. 3. Значит, можно получить быстрый способ проверки прародства, зная расположение каждого узла в прямом и обратном порядках обхода.)

21. Если узел NODE(P) является бинарным оператором, то указателями двух его операндов будут P1 = LLINK(P) *и* P2 = RLINK(P1) = $P. *В алгоритме D используется тот факт, что* P2$ = P, так что RLINK(P1) можно заменить указателем Q1, т. е. указателем на производную узла NODE(P1), а затем вновь переустановить RLINK(P1) на шаге D3. При обработке тернарных операций получим, что P1 = LLINK(P), P2 = RLINK(P1), P3 = RLINK(P2) = $P, а потому обобщить обработку бинарных операций довольно трудно. После вычисления производной Q1 можно временно установить RLINK(P1) $\leftarrow$ Q1, а затем после вычисления следующей производной Q2 установить RLINK(Q2) $\leftarrow$ Q1 и RLINK(P2) $\leftarrow$ Q2 и переустановить RLINK(P1) $\leftarrow$ P2. Но такое решение вряд ли можно считать элегантным, а с возрастанием степени операторов оно становится все более неуклюжим. Следовательно, способ на основе временного изменения RLINK(P1) в алгоритме D может рассматриваться всего лишь как *уловка*, но никак не как общий *метод* решения подобных задач. Более эстетичный способ управления процессом дифференцирования основан на алгоритме 2.3.3F (см. упр. 2.3.3–3), потому что он сформулирован в более общем виде так, что его можно применять для операторов с более высокими степенями и не прибегать к каким-либо трюкам.

22. Из данного определения сразу же следует, что такое отношение является транзитивным, т. е. если $T \subseteq T'$ и $T' \subseteq T''$, то $T \subseteq T''$. (На самом деле легко видеть, что это отношение является частичным упорядочением.) Если допустить, что f может отобразить каждый узел на себя, то ясно, что $l(T) \subseteq T$ и $r(T) \subseteq T$. Следовательно, если $T \subseteq l(T')$ или $T \subseteq r(T')$, то обязательно $T \subseteq T'$.

Предположим, что f_l и f_r являются функциями, для которых выполняются отношения $l(T) \subseteq l(T')$ и $r(T) \subseteq r(T')$ соответственно. Пусть $f(u) = f_l(u)$, если u принадлежит $l(T)$, $f(u) =$ корень дерева T', если u является корнем дерева T, а в противном случае $f(u) = f_r(u)$. Тогда для такой функции f выполняется отношение $T \subseteq T'$. Например, пусть $r'(T)$ обозначает $r(T)\backslash$корень дерева T. Тогда получим следующее: прямой порядок дерева $T =$ корень дерева T прямой порядок дерева $(l(T))$ прямой порядок дерева $(r'(T))$;

а прямой порядок дерева $T' = f(\text{корень дерева } T)$ прямой порядок дерева $(l(T'))$ прямой порядок дерева $(r'(T'))$.

Обратное утверждение не верно: рассмотрите поддеревья с корнями b и b' на рис. 25.

РАЗДЕЛ 2.3.3

1. Да, их можно восстановить точно так, как (3) можно вывести из (4), но заменяя LTAG и RTAG, а также LLINK и RLINK и используя очередь вместо стека.

2. В алгоритм F внесем следующие изменения. На шаге F1 фразу "P пусть указывает на первый узел леса в обратном порядке" заменим фразой "P пусть указывает на последний узел леса в прямом порядке". На шаге F2 в двух местах заменим фрагмент "$f(x_d), \ldots, f(x_1)$" фрагментом "$f(x_1), \ldots, f(x_d)$". На шаге F4 выполним такие действия: "Если P указывает на первый узел в прямом порядке, то прекратить выполнение алгоритма. (Тогда стек в направлении сверху вниз будет содержать элементы $f(\text{корень}(T_1))$, ..., $f(\text{корень}(T_m))$, где $T_1, \ldots, T_m$ — деревья данного леса по направлению слева направо.) В противном случае установить P указывающим на его предшественника в прямом порядке (P $\leftarrow$ P $-$ *c* для данного представления) и вернуться к шагу F2".

3. На шаге D1 следует установить S $\leftarrow \Lambda$. (S — переменная связи, которая указывает на верх стека.) Шаг D2 можно изменить таким образом: "Если NODE(P) обозначает унарный оператор, установить Q $\leftarrow$ S, S $\leftarrow$ RLINK(Q), P1 $\leftarrow$ LLINK(P). Если NODE(P) обозначает бинарный оператор, установить Q $\leftarrow$ S, Q1 $\leftarrow$ RLINK(Q), S $\leftarrow$ RLINK(Q1), P1 $\leftarrow$ LLINK(P), P2 $\leftarrow$ RLINK(P1). Затем выполнить DIFF[TYPE(P)]". Шаг D3 после изменений может выглядеть так: "Установить RLINK(Q) $\leftarrow$ S, S $\leftarrow$ Q", а шаг D4 будет таким: "Установить P $\leftarrow$ P$". Операцию LLINK(DY) $\leftarrow$ Q можно исключить на шаге D5, если предположить, что S $\equiv$ LLINK(DY). Очевидно, что этот метод можно обобщить для тернарных операторов и операторов более высокого порядка.

4. Для представления наподобие (10) потребуется $n - m$ полей LLINK и $n + (n - m)$ полей RLINK. Разница в общем количестве связей для этих двух типов представления равна $n - 2m$. Если поля LLINK и INFO используют примерно одно и то же пространство внутри узла, а m очень велико (а именно, если неконцевые узлы имеют очень высокую степень), представление (10) выглядит предпочтительнее.

5. Совершенно неразумно было бы включать прошитые связи RLINK, так как связь RLINK в любом случае указывает только на PARENT. Прошитые связи LLINK, подобные тем, которые использовались в 2.3.2–(4), были бы полезны только при обходе дерева справа налево, например если бы потребовалось совершить обход дерева в реверсивном обратном или фамильном порядке. Но эти операции без прошитых связей LLINK не так уж сложно выполнить, если только узлы имеют не очень высокую степень.

6. **L1.** Установить P $\leftarrow$ FIRST, FIRST $\leftarrow \Lambda$.

L2. Если P $= \Lambda$, прекратить выполнение алгоритма. В противном случае установить Q $\leftarrow$ RLINK(P).

L3. Если PARENT(P) $= \Lambda$, установить RLINK(P) $\leftarrow$ FIRST, FIRST $\leftarrow$ P. В противном случае установить R $\leftarrow$ PARENT(P), RLINK(P) $\leftarrow$ LCHILD(R), LCHILD(R) $\leftarrow$ P.

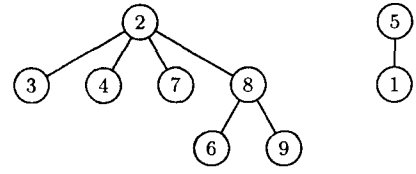
L4. Установить P $\leftarrow$ Q и вернуться к шагу L2. ▮

7. $\{1, 5\}\{2, 3, 4, 7\}\{6, 8, 9\}$.

8. Следует выполнить шаг E3 из алгоритма E, а затем проверить справедливость выражения $j = k$.

9.

PARENT[k]:	5	0	2	2	0	8	2	2	8
k :	1	2	3	4	5	6	7	8	9



10. Один из способов заключается в занесении в поле PARENT каждого *корневого* узла числа узлов его дерева со знаком "минус" (эти значения можно легко обновлять). Затем, если $|\texttt{PARENT}[j]| > |\texttt{PARENT}[k]|$ на шаге E4, происходит взаимный обмен ролями значений j и k. Как показал М. Д. Мак-Илрой (M. D. McIlroy), этот метод позволяет сократить количество шагов до $O(\log n)$.

Для достижения еще более высокой скорости можно использовать следующее предположение Алана Триттера (Alan Tritter): на шаге E4 установить $\texttt{PARENT}[x] \leftarrow k$ для всех значений $x \neq k$, которые встретились на шаге E3. Это вынуждает выполнить еще один проход вверх по дереву, но приводит также к сокращению их длины, а потому последующий поиск выполняется гораздо быстрее (см. раздел 7.4.1).

11. Достаточно определить преобразование, которое выполняется для каждого входного потока $(\texttt{P}, j, \texttt{Q}, k)$.

T1. *Если* $\texttt{PARENT(P)} \neq \Lambda$, *установить* $j \leftarrow j + \texttt{DELTA(P)}$, $\texttt{P} \leftarrow \texttt{PARENT(P)}$ *и повторить* этот шаг.

T2. Если $\texttt{PARENT(Q)} \neq \Lambda$, установить $k \leftarrow k + \texttt{DELTA(Q)}$, $\texttt{Q} \leftarrow \texttt{PARENT(Q)}$ и повторить этот шаг.

T3. Если $\texttt{P} = \texttt{Q}$, проверить справедливость условия $j = k$ (если оно не выполняется, значит, входной поток содержит противоречивые соотношения эквивалентности). Если $\texttt{P} \neq \texttt{Q}$, установить $\texttt{DELTA(Q)} \leftarrow j - k$, $\texttt{PARENT(Q)} \leftarrow \texttt{P}$, $\texttt{LBD(P)} \leftarrow \min(\texttt{LBD(P)}, \texttt{LBD(Q)} + \texttt{DELTA(Q)})$ и $\texttt{UBD(P)} \leftarrow \max(\texttt{UBD(P)}, \texttt{UBD(Q)} + \texttt{DELTA(Q)})$. ▮

Замечание. Для должным образом сформулированных условий можно допустить, чтобы объявления "ARRAY X[l:u]" поступали вперемешку с отношениями эквивалентности, либо допустить присвоение некоторых адресов переменным до того, как будет установлена их эквивалентность другим переменным, и т. д. Иные способы развития этого алгоритма можно найти в *CACM* **7** (1964), 301–303, 506.

12. (a) Да. (Если это условие не является обязательным, то можно избежать циклического повторения операций с S на шагах A2 и A9.) (b) Да.

13. Ключевым фактом является то, что направленная вверх от P цепь связей UP всегда охватывает те же переменные и степени этих переменных, что и направленная вверх от Q цепь связей UP, но в последней цепочке могут присутствовать дополнительные шаги для переменных со степенью "нуль". (Это условие выполняется на протяжении всего алгоритма за исключением шагов A9 и A10.) Теперь переход к шагу A8 возможен либо после шага A3, либо после шага A10, и в каждом таком случае выполняется проверка истинности условия $\texttt{EXP(Q)} \neq 0$. Следовательно, $\texttt{EXP(P)} \neq 0$. В частности, отсюда следует, что $\texttt{P} \neq \Lambda$, $\texttt{Q} \neq \Lambda$, $\texttt{UP(P)} \neq \Lambda$, $\texttt{UP(Q)} \neq \Lambda$, а значит, следует и справедливость сформулированного в упражнении утверждения. Таким образом, чтобы получить это доказательство, нужно показать, что указанное выше условие для цепочки связей UP при выполнении данного алгоритма не нарушается.

16. Докажите (методом индукции по количеству узлов *одного дерева* T), что если P указывает на T и если стек в исходном состоянии пуст, то после выполнения шагов F2–F4 он будет содержать только одно значение — $f(\text{корень}(T))$. Это верно для $n = 1$. Если $n > 1$, существуют $0 < d = \texttt{DEGREE}(\text{корень}(T))$ поддеревьев $T_1, \ldots, T_d$. По индукции и определению стека, а также на основании того, что при обратном порядке обхода после

$T_1, \ldots, T_d$ располагается корень дерева T, получим в результате применения этого алгоритма $f(T_1), \ldots, f(T_d)$, а затем и $f(\text{корень}(T))$, что и требовалось доказать. Аналогично можно доказать корректность алгоритма F для лесов.

17. G1. Опустошить стек и установить P указывающим на корень данного дерева (последний узел в обратном порядке). Вычислить $f(\texttt{NODE(P)})$.

G2. Поместить в стек DEGREE(P) копий $f(\texttt{NODE(P)})$.

G3. Если P является первым узлом в обратном порядке, прекратить выполнение алгоритма. В противном случае установить P указывающим на его предшественника в обратном порядке (например, для (9) это значит просто установить $\texttt{P} \leftarrow \texttt{P} - c$).

G4. Вычислить $f(\texttt{NODE(P)})$ с помощью значения верхнего элемента стека, которое равно $f(\texttt{NODE(PARENT(P))})$. Удалить это значение из стека и вернуться к шагу G2. ▮

Замечание. Алгоритм, аналогичный данному, может быть основан не на обратном, а на прямом порядке, как в упр. 2. На самом деле для этого можно использовать также фамильный порядок или порядок уровней, причем в последнем случае вместо стека можно использовать очередь.

18. Таблицы INFO1 и RLINK, а также описанный в данном разделе способ вычисления LTAG позволяют получить эквивалентное бинарное дерево в обычном представлении. Теперь для достижения цели нужно совершить обход этого дерева в обратном порядке, попутно подсчитывая степени.

P1. Пусть R, D и I — стеки, которые в исходном состоянии пусты. Тогда следует установить $\texttt{R} \Leftarrow n+1$, $\texttt{D} \Leftarrow 0$, $j \leftarrow 0$, $k \leftarrow 0$.

P2. Если (верх)R $> j+1$, перейти к шагу P5. (Если бы существовало поле LTAG, можно было бы вместо этого условия использовать условие $\texttt{LTAG}[j] = 0$.)

P3. Если стек I пуст, прекратить выполнение алгоритма; в противном случае установить $i \Leftarrow \texttt{I}$, $k \leftarrow k+1$, $\texttt{INFO2}[k] \leftarrow \texttt{INFO}[i]$, $\texttt{DEGREE}[k] \Leftarrow \texttt{D}$.

P4. Если $\texttt{RLINK}[i] = 0$, перейти к шагу P3; в противном случае удалить верхний элемент стека R (который равен $\texttt{RLINK}[i]$).

P5. Установить верх(D) $\leftarrow$ верх(D) $+ 1$, $j \leftarrow j+1$, $\texttt{I} \Leftarrow j$, $\texttt{D} \Leftarrow 0$. Если $\texttt{RLINK}[j] \neq 0$, установить $\texttt{R} \Leftarrow \texttt{RLINK}[j]$ и перейти к шагу P2. ▮

19. (a) Это эквивалентно утверждению, что связи SCOPE не пересекаются. (b) Первое дерево леса содержит $d_1 + 1$ элементов. Доказательство можно продолжить методом индукции. (c) Условие (a) выполняется для минимальных значений.

Замечания. Из упр. 2.3.2–20 следует, что $d_1 d_2 \ldots d_n$ также может интерпретироваться на основании инверсии: если k-й узел в обратном порядке является p_k-м узлом в прямом порядке, то d_k — это количество элементов $> k$, которые располагаются слева от k в $p_1 p_2 \ldots p_n$.

Аналогичная схема, в которой перечисляется количество наследников каждого узла в *обратном* порядке для данного леса, приводит к последовательности чисел $c_1 c_2 \ldots c_n$, которые характеризуются свойствами (i) $0 \le c_k < k$ и (ii) $k \ge j \ge k - c_k$, из чего следует $j - c_j \ge k - c_k$. Алгоритмы на основе таких последовательностей были исследованы Ж. М. Палло (J. M. Pallo) в работе *Comp. J.* **29** (1986), 171–175. Обратите внимание, что c_k является размером левого поддерева k-го узла в симметричном порядке обхода соответствующего бинарного дерева. d_k также можно интерпретировать как размер *правого* поддерева k-го узла в симметричном порядке обхода соответствующего бинарного дерева, а именно — бинарного дерева, которое соответствует данному лесу согласно методу из упр. 2.3.2–5.

Отношение $d_k \le d'_k$ для $1 \le k \le n$ определяет интересное решеточное упорядочение лесов и бинарных деревьев, которое впервые было предложено Д. Тамари (D. Tamari) [Thèse (Paris, 1951)] с помощью другого способа (см. упр. 6.2.3–32).

РАЗДЕЛ 2.3.4.1

1. (B, A, C, D, B), (B, A, C, D, E, B), (B, D, C, A, B), (B, D, E, B), (B, E, D, B), (B, E, D, C, A, B).

2. Пусть $(V_0, V_1, \ldots, V_n)$ — путь с минимально возможной длиной от вершины V к вершине V'. Если бы $V_j = V_k$ для некоторого $j < k$, то путь $(V_0, \ldots, V_j, V_{k+1}, \ldots, V_n)$ был бы еще короче.

3. (Фундаментальный путь проходит один раз через ребра e_3 и e_4, а цикл C_2 проходит через них -1 раз, что в сумме дает нуль.) Путь проходит через такие ребра: e_1, e_2, e_6, e_7, e_9, e_{10}, e_{11}, e_{12}, e_{14}.

4. Если утверждение задачи не выполняется, то допустим, что G'' является таким подграфом для графа G', который получен в результате удаления каждого ребра e_j, для которого $E_j = 0$. Тогда G'' является конечным графом без циклов и с минимум одним ребром. Поэтому согласно теореме A существует по крайней мере одна такая вершина V, что она является смежной в точности для одной вершины V'. Пусть e_j — ребро, соединяющее вершины V и V', тогда уравнение Кирхгофа (1) для вершины V выглядит как $E_j = 0$, что противоречит определению G''.

5. $A = 1 + E_8$, $B = 1 + E_8 - E_2$, $C = 1 + E_8$, $D = 1 + E_8 - E_5$, $E = 1 + E_{17} - E_{21}$, $F = 1 + E''_{13} + E_{17} - E_{21}$, $G = 1 + E''_{13}$, $H = E_{17} - E_{21}$, $J = E_{17}$, $K = E''_{19} + E_{20}$, $L = E_{17} + E''_{19} + E_{20} - E_{21}$, $P = E_{17} + E_{20} - E_{21}$, $Q = E_{20}$, $R = E_{17} - E_{21}$, $S = E_{25}$. *Замечание.* В этом случае с помощью переменных $A, B, \ldots, S$ можно выразить также значения $E_2, E_5, \ldots, E_{25}$. Следовательно, имеется девять независимых решений, и именно поэтому из упр. 1.3.3–(8) были исключены шесть переменных.

6. (Следующее решение основано на идее о том, что распечатывать можно каждое ребро, которое не образует цикл с предыдущими ребрами.) Используйте алгоритм 2.3.3E, в котором каждая пара (a_i, b_i) представляет *соотношение* $a_i \equiv b_i$ в обозначениях этого алгоритма. Единственное внесенное изменение заключается в печати пар (a_i, b_i), если $j \neq k$ на шаге E4.

Чтобы показать справедливость данного алгоритма, нужно доказать, что (a) алгоритм не выводит на печать ребра, которые образуют цикл, (b) если граф G содержит по крайней мере одно свободное поддерево, то данный алгоритм распечатает $n - 1$ ребер. Определим отношение $j \equiv k$, если существует путь от вершины V_j до вершины V_k или если соблюдается условие $j = k$. Очевидно, что это отношение эквивалентности, причем $j \equiv k$ тогда и только тогда, когда оно может быть выведено из отношений эквивалентности $a_1 \equiv b_1, \ldots, a_m \equiv b_m$. Утверждение (a) справедливо, потому что алгоритм не выводит на печать ребра, которые образуют цикл с прежде распечатанными ребрами, а утверждение (b) справедливо, потому что `PARENT[`k`]` $= 0$ только для одного k, если все вершины являются эквивалентными.

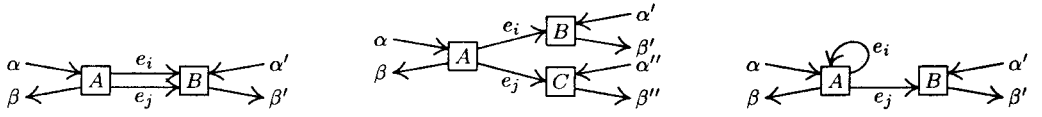
Более эффективный алгоритм, однако, должен быть основан на поиске в глубину (см. алгоритм 2.3.5A и раздел 7.4.1).

7. Фундаментальные циклы: $C_0 = e_0 + e_1 + e_4 + e_9$ (фундаментальным путем будет $e_1 + e_4 + e_9$); $C_5 = e_5 + e_3 + e_2$; $C_6 = e_6 - e_2 + e_4$; $C_7 = e_7 - e_4 - e_3$; $C_8 = e_8 - e_9 - e_4 - e_3$. На основании этого получим $E_1 = 1$, $E_2 = E_5 - E_6$, $E_3 = E_5 - E_7 - E_8$, $E_4 = 1 + E_6 - E_7 - E_8$, $E_9 = 1 - E_8$.

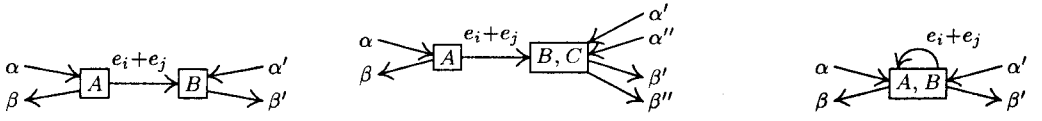
8. На каждом шаге процедуры приведения объединяются две стрелки e_i и e_j, которые выходят из одного и того же блока, а потому достаточно доказать, что эти действия можно обратить. Таким образом, если состояние ребер $e_i + e_j$ после их объединения известно, необходимо указать состояние для ребер e_i и e_j до объединения. При этом могут существовать три разных случая.

Случай 1 Случай 2 Случай 3

Состояние до



Состояние после



Здесь A, B и C — вершины или супервершины, а α и β — другие потоки, отличные от потока $e_i + e_j$. Эти потоки могут быть распределены среди нескольких ребер, хотя здесь показана только одна вершина. В случае 1 (ребра e_i и e_j направлены к одному и тому же блоку) можно произвольным образом выбрать ребро e_i, и тогда $e_j \leftarrow (e_i + e_j) - e_i$. В случае 2 ($e_i$ и e_j направлены к разным блокам) обязательно получим $e_i \leftarrow \beta' - \alpha'$, $e_j \leftarrow \beta'' - \alpha''$. В случае 3 (ребро e_i является циклом, а e_j — нет) обязательно получим $e_j \leftarrow \beta' - \alpha'$, $e_i \leftarrow (e_i + e_j) - e_j$. Таким образом, процедура объединения обращена во всех трех случаях.

Результат выполнения этого упражнения доказывает, что количество фундаментальных циклов в приведенной блок-схеме равно минимальному количеству потоков вершин, измерив которые, можно определить все потоки. В рассматриваемом примере приведенной блок-схемы достаточно вычислить только три потока вершин (например, a, c, d), тогда как исходная схема из упр. 7 имеет четыре независимых потока ребер. Всякий раз, когда возникает случай 1, одно измерение можно сэкономить.

Аналогичная процедура приведения может быть основана на объединении стрелок, которые направлены *на* данный блок, а не из него. Можно показать, что подобным образом получаем такую же приведенную блок-схему, но с другими именами супервершин.

Данное построение основано на идеях Армена Нахапетяна (Armen Nahapetian) и Ф. Стивенсона (F. Stevenson). Более подробно этот вопрос рассматривается в работе [D. E. Knuth and F. Stevenson, *BIT* **13** (1973), 313–322].

9. Каждое ребро, выходящее из вершины и сразу же замыкающееся на ней, называется совершенно отдельным фундаментальным циклом. Если существует $k + 1$ ребер $e, e', \ldots, e^{(k)}$ между вершинами V и V', построим k фундаментальных циклов $e' \pm e$, $\ldots$, $e^{(k)} \pm e$ (выбирая "+" или "−" в зависимости от того, совпадает ли направление обхода ребра с направлением стрелки), а затем продолжим работу так, как будто существует только ребро e.

Действительно, эту ситуацию можно существенно упростить, если определить граф таким образом, чтобы в нем допускалось наличие нескольких ребер между одинаковыми вершинами, а также определить ребра, концы которых могут замыкаться на одном узле, причем пути и циклы определять на основе ребер, а не вершин. Такая формулировка позволяет дать определение ориентированного дерева, которое приводится в разделе 2.3.4.2.

10. Если все выводы соединены в единую сеть, то соответствующий ей граф должен быть связным. При этом сеть с минимальным количеством соединений не будет включать циклов, а потому получится свободное дерево. По теореме А свободное дерево содержит $n-1$ соединений, а граф с n вершинами и $n-1$ ребрами является свободным деревом тогда и только тогда, когда он является связным.

11. Достаточно доказать, что когда $n > 1$ и минимальное значение $c(i, n)$ равно $c(n-1, n)$, то существует по крайней мере одно дерево с минимальной ценой, в котором вывод T_{n-1} соединен с T_n. (Действительно, для любого дерева с минимальной ценой, в котором имеется $n > 1$ концевых узлов, и с выводом T_{n-1}, соединенным с выводом T_n, должно существовать и дерево с минимальной ценой с $n-1$ концевыми узлами, если рассматривать выводы T_{n-1} и T_n как "один обобщенный вывод", используя упомянутое в этом алгоритме соглашение.)

Для доказательства приведенного выше выражения предположим, что имеется дерево с минимальной ценой, в котором вывод T_{n-1} "не спаян" с T_n. Если добавить связующее звено T_{n-1} — T_n, то получится цикл, причем в нем можно удалить любое другое связующее звено. Удалив связующее звено, которое касается вывода T_n, получим другое дерево, общая цена которого не выше исходной цены и в котором содержится связующее звено T_{n-1} — T_n.

12. Создайте две вспомогательные таблицы $a(i)$ и $b(i)$ для $1 \le i < n$, в которых $T_{b(i)}$ — самое дешевое соединение между выводом T_i и выбранным выводом, а $a(i)$ — его стоимость. В исходном состоянии $a(i) = c(i, n)$ и $b(i) = n$. Затем выполните следующие операции $n-1$ раз: найдите такое значение i, что $a(i) = \min_{1 \le j < n} a(j)$; соедините T_i с $T_{b(i)}$; для $1 \le j < n$, если $c(i, j) < a(j)$, установите $a(j) \leftarrow c(i, j)$ и $b(j) \leftarrow i$; установите $a(i) \leftarrow \infty$. Здесь $c(i, j)$ означает $c(j, i)$, когда $j < i$.

(В определенной степени эффективнее было бы использовать не ∞, а однонаправленный связанный список для таких j, которые еще не были отобраны. С учетом или без учета этого прямолинейного подхода для выполнения алгоритма потребуется около $O(n^2)$ операций.) См. также работы E. W. Dijkstra, *Proc. Nederl. Akad. Wetensch.* **A63** (1960), 196–199; D. E. Knuth, *The Stanford GraphBase* (New York: ACM Press, 1994), 460–497. Более совершенные алгоритмы поиска дерева с минимальной ценой рассматриваются в разделе 7.5.4.

13. Если не существует никакого пути от вершины V_i до вершины V_j для некоторого $i \ne j$, то никакое произведение транспозиций не сможет переместить i на место j. Поэтому, если генерируются все перестановки, граф должен быть связным. И наоборот, если граф является связным, при необходимости удаляйте ребра до тех пор, пока не получите дерево. Затем перенумеруйте вершины так, чтобы вершина V_n была смежной только для одной другой вершины, а именно — для V_{n-1} (см. доказательство теоремы A). Теперь все транспозиции, кроме $(n-1\ n)$, образуют дерево с $n-1$ вершинами. Поэтому по методу индукции, если π является перестановкой $\{1, 2, \ldots, n\}$ с фиксированным расположением n, π можно представить в виде произведения этих транспозиций. Если π перемещает n в положение j, то $\pi(j\ n-1)(n-1\ n) = \rho$ фиксирует положение n. Следовательно, $\pi = \rho(n-1\ n)(j\ n-1)$ можно представить как произведение заданных транспозиций.

РАЗДЕЛ 2.3.4.2

1. Пусть $(e_1, \ldots, e_n)$ — ориентированный путь от V к V' с минимально возможной длиной. Если теперь $\text{init}(e_j) = \text{init}(e_k)$ для $j < k$, то $(e_1, \ldots, e_{j-1}, e_k, \ldots, e_n)$ будет кратчайшим путем, и аналогичный аргумент можно использовать, если $\text{fin}(e_j) = \text{fin}(e_k)$ для $j < k$. Следовательно, путь $(e_1, \ldots, e_n)$ является простым.

2. Те циклы, в которых все знаки одинаковы: C_0, C_8, C''_{13}, C_{17}, C''_{19}, C_{20}.

3. Для этого, например, можно использовать три вершины A, B и C с дугами от A к B и от A к C.

4. Если в графе G нет ориентированных циклов, алгоритм 2.2.3T позволяет выполнить его топологическую сортировку. А если ориентированный цикл существует, топологиче-

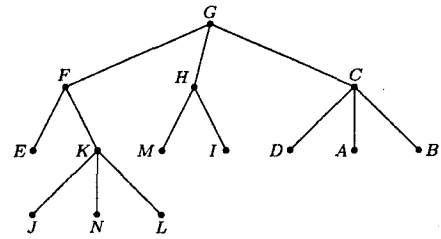
скую сортировку, очевидно, выполнить нельзя. (В зависимости от интерпретации этого упражнения ориентированные циклы с длиной 1 можно было бы вообще не рассматривать.)

5. Пусть k — такое наименьшее целое число, что $\mathrm{fin}(e_k) = \mathrm{init}(e_j)$ для некоторого $j \le k$. Тогда $(e_j, \ldots, e_k)$ — ориентированный цикл.

6. Нет (формально говоря) именно потому, что может существовать несколько различных дуг от одной вершины к другой.

7. Да, оно справедливо для конечных ориентированных графов. Если начать с произвольной вершины V и совершать обход только единственно возможного ориентированного пути, то ни одна вершина на этом пути никогда не встретится дважды. В конце концов мы обязательно достигнем вершины R (т. е. единственной вершины без наследника). Для бесконечных ориентированных графов это утверждение, очевидно, неверно, так как в нем могут быть вершины $R, V_1, V_2, V_3, \ldots$ и дуги от V_j к V_{j+1} для $j \ge 1$.

9. Все дуги направлены вверх.



10. **G1.** Установить $k \leftarrow P[j]$, $P[j] \leftarrow 0$.

G2. Если $k = 0$, прекратить выполнение алгоритма; в противном случае установить $m \leftarrow P[k]$, $P[k] \leftarrow j$, $j \leftarrow k$, $k \leftarrow m$ и повторить шаг G2. ▮

11. *Данный алгоритм основан на алгоритме* 2.3.3E *и методе, описанном в предыдущем* упражнении, поэтому все ориентированные деревья имеют дуги, которые соответствуют дугам в ориентированном графе. Здесь $S[j]$ — вспомогательная таблица, в которой указано направление дуги либо от j к $P[j]$ ($S[j] = +1$), либо от $P[j]$ к j ($S[j] = -1$). В исходном состоянии $P[1] = \cdots = P[n] = 0$. Для обработки каждой дуги (a, b) можно выполнить такие действия.

C1. Установить $j \leftarrow a$, $k \leftarrow P[j]$, $P[j] \leftarrow 0$, $s \leftarrow S[j]$.

C2. Если $k = 0$, перейти к шагу C3; в противном случае установить $m \leftarrow P[k]$, $t \leftarrow S[k]$, $P[k] \leftarrow j$, $S[k] \leftarrow -s$, $s \leftarrow t$, $j \leftarrow k$, $k \leftarrow m$ и повторить шаг C2.

C3. (Теперь a становится корнем этого дерева.) Установить $j \leftarrow b$, а затем, если $P[j] \neq 0$, повторно задавать значение $j \leftarrow P[j]$ до тех пор, пока не выполнится условие $P[j] = 0$.

C4. Если $j = a$, перейти к шагу C5; в противном случае установить $P[a] \leftarrow b$, $S[a] \leftarrow +1$, распечатать (a, b) как дугу, которая относится к свободному поддереву, и прекратить выполнение алгоритма.

C5. Напечатать сначала "`CYCLE`", а затем — "(a, b)".

C6. Если $P[b] = 0$, прекратить выполнение алгоритма. В противном случае, если $S[b] = +1$, напечатать "$+(b, P[b])$", а если нет, то напечатать "$-(P[b], b)$"; установить $b \leftarrow P[b]$ и повторить шаг C6. ▮

Замечание. Если использовать допущение Мак-Илроя из ответа к упр. 2.3.3–10, для выполнения этого алгоритма потребуется осуществить $O(m \log n)$ этапов. Но есть и более удачное решение, для которого потребуется выполнить только $O(m)$ этапов. Используйте поиск преимущественно в глубину для построения "пальмы" (palm tree) с одним фундаментальным циклом для каждого "листа" [R. E. Tarjan, *SICOMP* **1** (1972), 146–150].

12. Степень узла дерева равна степени входа, а степень выхода каждой вершины может быть равна только 0 или 1.

13. Определим последовательность ориентированных поддеревьев графа G следующим образом: G_0 содержит лишь одну вершину R; G_{k+1} содержит G_k и любую вершину V графа G, не принадлежащую графу G_k, но для которой существует дуга от V к V', где V' *принадлежит* графу G_k, а также еще одна такая дуга $e[V]$ для каждой такой вершины. Отсюда по индукции немедленно следует, что G_k — ориентированное дерево для всех $k \geq 0$ и что если есть ориентированный путь длиной k от V к R в G, то V принадлежит графу G_k. Следовательно, G_∞ — это множество всех V и $e[V]$ в любом G_k и оно является искомым ориентированным поддеревом графа G.

14. В лексикографическом порядке они выглядят следующим образом:
$(e_{12}, e_{20}, e_{00}, e'_{01}, e_{10}, e_{01}, e'_{12}, e_{22}, e_{21})$, $(e_{12}, e_{20}, e_{00}, e'_{01}, e'_{12}, e_{22}, e_{21}, e_{10}, e_{01})$,
$(e_{12}, e_{20}, e'_{01}, e_{10}, e_{00}, e_{01}, e'_{12}, e_{22}, e_{21})$, $(e_{12}, e_{20}, e'_{01}, e'_{12}, e_{22}, e_{21}, e_{10}, e_{00}, e_{01})$,
$(e_{12}, e_{22}, e_{20}, e_{00}, e'_{01}, e_{10}, e_{01}, e'_{12}, e_{21})$, $(e_{12}, e_{22}, e_{20}, e_{00}, e'_{01}, e'_{12}, e_{21}, e_{10}, e_{01})$,
$(e_{12}, e_{22}, e_{20}, e'_{01}, e_{10}, e_{00}, e_{01}, e'_{12}, e_{21})$, $(e_{12}, e_{22}, e_{20}, e'_{01}, e'_{12}, e_{21}, e_{10}, e_{00}, e_{01})$.

Восемь вариантов получены в результате перебора независимых пар ребер, в каждой из которых одно из ребер может предшествовать другому: e_{00} или e'_{01}, e_{10} или e'_{12}, e_{20} или e_{22}.

15. Да, справедливо, так как если связный и сбалансированный граф имеет более одной вершины, то он содержит цепь Эйлера, которая проходит через все вершины.

16. Рассмотрим ориентированный граф G с вершинами $V_1, \ldots, V_{13}$ и с дугой от V_j к V_k для каждого k в стопке j. Выигрыш такой игры эквивалентен обходу цепи Эйлера в этом ориентированном графе (действительно, если игра является выигрышной, то заключительная перевернутая карта должна быть взята из центральной стопки; следовательно, этот граф является сбалансированным). Теперь, если игра выигрышная, указанный диграф является ориентированным поддеревом согласно лемме E. И наоборот, если указанный диграф является ориентированным деревом, то по теореме D игра является выигрышной.

17. $\frac{1}{13}$. Такой ответ можно получить посредством трудоемкого перечисления ориентированных деревьев особого типа, применения производящих функций и т. д., основанных на результатах из раздела 2.3.4.4. (Именно таким образом автор впервые получил данный результат.) Кроме того, его можно получить из следующего очень простого прямого доказательства. Определим порядок переворачивания *всех* карт из колоды. Будем раскладывать пасьянс по приведенным выше правилам до тех пор, пока ситуация не станет неразрешимой, а затем смошенничаем, перевернув первую доступную карту (найдем первую стопку, которая не является пустой, передвигаясь по часовой стрелке от стопки 1) и продолжив раскладывать пасьянс, как и прежде, до тех пор, пока в конце концов не будут перевернуты все карты. Карты *при таком порядке переворачивания* будут упорядочены совершенно случайным образом (поскольку значение карты потребуется узнать только после ее переворачивания). Поэтому проблема заключается только в подсчете вероятности того, что в полностью перетасованной колоде карт последней картой является "король". В более общей формулировке вероятность того, что k карт остаются повернутыми лицевой стороной вверх по завершении игры, равна вероятности того, что за последней картой с изображением короля в перетасованной колоде карт следует еще k карт, а именно — $4!\binom{51-k}{3}\frac{48!}{52!}$. Следовательно, играя честно, без мошенничества, в среднем за игру придется перевернуть в точности 42.4 карты. *Замечание.* Аналогично можно показать, что вероятность того, что игрок смошенничает k раз в ходе описанного выше процесса в точности равна числу Стирлинга $\left[{13 \atop k+1}\right]/13!$ (см. уравнение 1.2.10–(9) и упр. 1.2.10–7; более общий случай рассматривается в упр. 1.2.10–18).

18. (a) Если существует цикл $(V_0, V_1, \ldots, V_k)$, в котором обязательно $3 \leq k \leq n$, сумма k строк матрицы A, соответствующих k ребрам этого цикла с соответствующими знаками,

будет строкой нулей. Поэтому, если граф G не является свободным деревом, детерминант матрицы A_0 равен нулю.

Но, если граф G является свободным деревом, его можно рассматривать как упорядоченное дерево с корнем V_0, а строки и столбцы матрицы A_0 переупорядочить так, чтобы столбцы располагались в прямом порядке и k-я строка соответствовала ребру от k-й вершины (столбца) к ее родителю. Тогда матрица будет треугольной с элементами ±1 по диагонали и ее детерминант будет равен ±1.

(b) По формуле Бине-Коши (см. упр. 1.2.3–46) получим

$$\det A_0^T A_0 = \sum_{1\le i_1<\cdots<i_n\le m} (\det A_{i_1\ldots i_n})^2,$$

где $A_{i_1\ldots i_n}$ — матрица из строк $i_1,\ldots,i_n$ матрицы A_0 (таким образом, соответствующая некоторому выбору n ребер графа G). Тогда ответ следует из (a).

[См. S. Okada and R. Onodera, *Bull. Yamagata Univ.* **2** (1952), 89–117.]

19. (a) Условия $a_{00}=0$ и $a_{jj}=1$ представляют собой условия (a) и (b) из определения ориентированного дерева. Если G не является ориентированным деревом, то существует ориентированный цикл (согласно результату упр. 7), а строки матрицы A_0, соответствующие вершинам ориентированного цикла в сумме дадут строку нулей; следовательно, $\det A_0 = 0$. Если G — ориентированное дерево, зададим произвольный порядок для детей каждой семьи и рассмотрим G как упорядоченное дерево. Теперь будем переставлять строки и столбцы матрицы A_0 до тех пор, пока они не будут соответствовать прямому порядку вершин. Так как одна и та же перестановка применяется как к строкам, так и к столбцам, детерминант матрицы остается неизменным. Полученная в результате матрица является треугольной со значениями $+1$ по диагонали.

(b) Можно допустить, что $a_{0j}=0$ для всех j, так как выходящие из вершины V_0 дуги не могут входить в состав ориентированного поддерева. Можно также предположить, что $a_{jj}>0$ для всех $j\ge1$, поскольку в противном случае во всей j-й строке содержатся нули и ориентированных поддеревьев, очевидно, не существует. Далее воспользуемся методом индукции по количеству дуг. Если $a_{jj}>1$, то пусть e — некоторая дуга, выходящая из V_j; пусть B_0 — матрица, подобная A_0, но в которой удалена дуга e, и пусть C_0 — матрица, подобная A_0, но в которой удалены все дуги, *за исключением* дуги e, выходящей из вершины V_j. *Пример.* Если $A_0=\left(\begin{smallmatrix}3&-2\\-1&2\end{smallmatrix}\right)$, $j=1$ и e — дуга от V_1 к V_0, то $B_0=\left(\begin{smallmatrix}2&-2\\-1&2\end{smallmatrix}\right)$, $C_0=\left(\begin{smallmatrix}1&0\\-1&2\end{smallmatrix}\right)$. В целом, получим $\det A_0=\det B_0+\det C_0$, так как в этих матрицах совпадают все строки, за исключением j-й строки, а j-я строка матрицы A_0 равна сумме j-х строк матриц B_0 и C_0. Более того, количество ориентированных поддеревьев графа G равно количеству поддеревьев, в которые *не* входит ребро e (а именно, по индукции оно равно $\det B_0$), плюс количество поддеревьев, в которые *входит* ребро e (а именно, $\det C_0$).

Замечания. Матрица A часто называется *лапласианом* (*Laplacian*) этого графа, по аналогии с подобным понятием из теории дифференциальных уравнений с частными производными. При удалении любого множества S строк из матрицы A и такого же множества столбцов детерминант полученной в итоге матрицы будет равен количеству ориентированных деревьев, корни которых являются вершинами $\{V_k \mid k\in S\}$ и дуги которых принадлежат данному диграфу. Теорема о матрице, соответствующей дереву, впервые была сформулирована без доказательства для ориентированных деревьев Дж. Дж. Сильвестром (J. J. Sylvester) в 1857 году (см. упр. 28), а затем надолго забыта, пока не была повторно исследована В. Т. Тутте (W. T. Tutte) [*Proc. Cambridge Phil. Soc.* **44** (1948), 463–482, §3]. Ее доказательство для особого случая *неориентированных* графов, когда матрица A является симметричной, было впервые опубликовано К. В. Борхардтом (C. W. Borchardt) [*Crelle* **57** (1860), 111–121]. Некоторые авторы приписывают доказательство этой теоремы Кирхгофу, но он доказал совершенно другой (хотя и близкий) результат.

20. Используя упр. 18, получим $B = A_0^T A_0$. Или, используя упр. 19, получим, что матрица B — это матрица A_0 ориентированного графа G', в которой каждое ребро заменено двумя дугами (по одной для каждого направления). Каждое свободное поддерево графа G соответствует единственному ориентированному поддереву графа G' с корнем V_0, так как направления дуг определяются выбором корня.

21. Постройте матрицы A и A^* так же, как в упр. 19. Например, для графов G и G^*, показанных на рис. 36 и 37, получим

$$A = \begin{pmatrix} 2 & -2 & 0 \\ -1 & 3 & -2 \\ -1 & -1 & 2 \end{pmatrix}, \qquad A^* = \begin{array}{c} \\ [00] \\ [10] \\ [20] \\ [01] \\ [01] \\ [21] \\ [12] \\ [12] \\ [22] \end{array} \begin{array}{c} \begin{array}{ccccccccc} [00] & [10] & [20] & [01] & [01] & [21] & [12] & [12] & [22] \end{array} \\ \begin{pmatrix} 2 & 0 & 0 & -1 & -1 & 0 & 0 & 0 & 0 \\ -1 & 3 & 0 & -1 & -1 & 0 & 0 & 0 & 0 \\ -1 & 0 & 3 & -1 & -1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 3 & 0 & 0 & -1 & -1 & 0 \\ 0 & -1 & 0 & 0 & 3 & 0 & -1 & -1 & 0 \\ 0 & -1 & 0 & 0 & 0 & 3 & -1 & -1 & 0 \\ 0 & 0 & -1 & 0 & 0 & -1 & 3 & 0 & -1 \\ 0 & 0 & -1 & 0 & 0 & -1 & 0 & 3 & -1 \\ 0 & 0 & -1 & 0 & 0 & -1 & 0 & 0 & 2 \end{pmatrix} \end{array}.$$

Сложите неопределенную величину λ с элементом в верхнем левом углу матриц A и A^* (в данном примере получим $2+\lambda$ вместо 2). Если $t(G)$ и $t(G^*)$ — количество ориентированных поддеревьев в графах G и G^* соответственно, то получим $t(G) = \lambda^{-1}(n+1)\det A$, $t(G^*) = \lambda^{-1}m(n+1)\det A^*$. (Количество ориентированных поддеревьев сбалансированного графа одинаково для любого выбора корня согласно упр. 22.)

Если сгруппировать вершины V_{jk} для равных k, то матрицу A^* можно разделить так, как показано выше. Пусть $B_{kk'}$ — подматрица матрицы A^*, которая состоит из строк для вершины V_{jk} и столбцов для вершины $V_{j'k'}$ для всех таких j и j', что V_{jk} и $V_{j'k'}$ принадлежат графу G^*. Складывая 2-й, ..., m-й столбец каждой подматрицы с первым столбцом, а затем вычитая первую строку каждой подматрицы из 2-й, ..., m-й строки, матрицу A^* можно привести к такому виду:

$$B_{kk'} = \begin{pmatrix} a_{kk'} & * & \dots & * \\ 0 & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 0 \end{pmatrix} \quad \text{для } k \neq k', \qquad B_{kk} = \begin{pmatrix} a_{kk} + \lambda\delta_{k0} & * & \dots & * \\ -\lambda\delta_{k0} & m & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ -\lambda\delta_{k0} & 0 & \dots & m \end{pmatrix}.$$

Отсюда следует, что $\det A^*$ в $m^{m(n-1)}$ раз больше детерминанта матрицы

$$\begin{pmatrix} \lambda + a_{00} & * & * & \dots & * & a_{01} & \dots & a_{0n} \\ -\lambda & m & 0 & \dots & 0 & 0 & \dots & 0 \\ \vdots & & & & & & & \\ -\lambda & 0 & 0 & \dots & m & 0 & \dots & 0 \\ a_{10} & * & * & \dots & * & a_{11} & \dots & a_{1n} \\ \vdots & \vdots & \vdots & \ddots & \vdots & & & \\ a_{n0} & * & * & \dots & * & a_{n1} & \dots & a_{nn} \end{pmatrix}.$$

В этих выкладках символ "$*$" обозначает величины, которые не имеют отношения к данной задаче, причем все они равны нулю, за исключением одной звездочки в каждом столбце, которая равна -1. Сложим последние n строк с верхней строкой и разложим детерминант по первой строке, чтобы получить $m^{n(m-1)+m-1}\det A - (m-1)m^{n(m-1)+m-2}\det A$.

Эти рассуждения можно обобщить для определения количества ориентированных поддеревьев графа G^*, когда граф G является *произвольным* ориентированным графом (см. R. Dawson and I. J. Good, *Ann. Math. Stat.* **28** (1957), 946–956; D. E. Knuth, *Journal*

of Combinatorial Theory **3** (1967), 309–314). Альтернативное и чисто комбинаторное доказательство предложено Дж. Б. Орлином [J. B. Orlin, *Journal of Combinatorial Theory* **B25** (1978), 187–198].

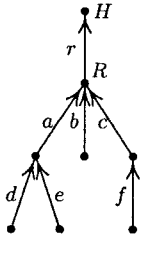
22. Общее количество цепей Эйлера равно числу $(\sigma_1 + \cdots + \sigma_n)$, которое умножено на количество цепей Эйлера, начинающихся с данного ребра e_1, где $\text{init}(e_1) = V_1$. Каждая такая цепь определяет ориентированное поддерево с корнем V_1 согласно лемме E, а для каждого из T ориентированных поддеревьев существует $\prod_{j=1}^{n}(\sigma_j - 1)!$ путей, которые удовлетворяют трем условиям теоремы D, соответствующим различному порядку входа дуг $\{e \mid \text{init}(e) = V_j,\ e \neq e[V_j],\ e \neq e_1\}$ в P. (В упр. 14 приводится простой пример такой ситуации.)

23. Постройте ориентированный граф G_k с m^{k-1} вершинами, как в указании, и используйте обозначение $[x_1, \ldots, x_k]$ для упомянутой в нем дуги. Для каждой функции, которая имеет максимальный период, можно определить единственную соответствующую цепь Эйлера, предполагая, что $f(x_1, \ldots, x_k) = x_{k+1}$, если за дугой $[x_1, \ldots, x_k]$ следует дуга $[x_2, \ldots, x_{k+1}]$. (Цепи Эйлера считаются одинаковыми, если одна из них является результатом циклической перестановки другой.) Теперь $G_k = G^*_{k-1}$ согласно обозначениям из упр. 21, поэтому G_k имеет в $m^{m^{k-1}-m^{k-2}}$ раз больше ориентированных поддеревьев, чем G_{k-1}. По индукции граф G_k имеет $m^{m^{k-1}-1}$ ориентированных поддеревьев и $m^{m^{k-1}-k}$ деревьев с заданным корнем. Следовательно, согласно ответу к упр. 22 количество функций с максимальным периодом, а именно — количество цепей Эйлера графа G_k, которые начинаются с заданной дуги, равно $m^{-k}(m!)^{m^{k-1}}$. [Для $m = 2$ этот результат приводится в работе C. Flye Sainte-Marie, L'*Intermédiaire des Mathématiciens* **1** (1894), 107–110.]

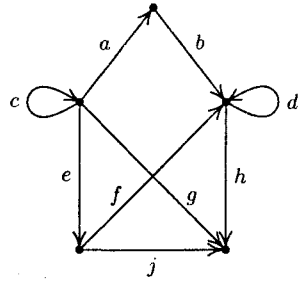
24. Определите новый ориентированный граф, который имеет E_j копий e_j для $0 \leq j \leq m$. Этот граф является сбалансированным, следовательно, по теореме G он содержит цепь Эйлера $(e_0, \ldots)$. Искомый ориентированный путь можно получить, удалив ребро e_0 из цепи Эйлера.

25. Зададим произвольный порядок для всех дуг в множествах $I_j = \{e \mid \text{init}(e) = V_j\}$ и $F_j = \{e \mid \text{fin}(e) = V_j\}$. Для каждой дуги e в I_j пусть ATAG$(e) = 0$ и ALINK$(e) = e'$, если e' расположено за e в упорядочении множества I_j, а также пусть ATAG$(e) = 1$ и ALINK$(e) = e'$, если e — последний элемент множества I_j и e' — первый элемент множества F_j. Пусть в последнем случае ALINK$(e) = \Lambda$, если множество F_j пусто. Определим BLINK и BTAG согласно тем же правилам, лишь меняя роли init и fin.

Примеры (каждый набор дуг расположен в алфавитном порядке).



Дуга	ALINK	ATAG	BLINK	BTAG
a	d	1	b	0
b	Λ	1	c	0
c	f	1	r	1
d	Λ	1	e	0
e	Λ	1	a	1
f	Λ	1	c	1
r	a	1	Λ	1



Дуга	ALINK	ATAG	BLINK	BTAG
a	c	0	b	1
b	a	1	d	0
c	e	0	a	1
d	h	0	f	0
e	g	0	f	1
f	j	0	d	1
g	c	1	h	0
h	b	1	j	0
j	e	1	Λ	1

Замечание. Если в представлении ориентированного дерева добавить другую дугу от H к ней самой, то возникнет интересная ситуация: будут получены либо стандартные условия 2.3.1–(8) с *взаимной заменой* полей LLINK, LTAG, RLINK, RTAG в заголовке списка, либо (если новая дуга располагается последней в данном упорядочении) стандартные условия, за исключением RTAG $= 0$ в узле, который соответствует корню этого дерева.

Настоящее упражнение основано на идее, которую автору сообщил В. Ч. Линч (W. C. Lynch). *Можно ли, используя такое представление, обобщить алгоритмы обхода деревьев, подобные алгоритму 2.3.1S, для классов диграфов, которые не являются ориентированными деревьями?*

27. Пусть a_{ij} — сумма вероятностей $p(e)$ для всех дуг e от V_i к V_j. Следует доказать, что $t_j = \sum_i a_{ij}t_i$ для всех j. Так как $\sum_i a_{ji} = 1$, необходимо доказать, что $\sum_i a_{ji}t_j = \sum_i a_{ij}t_i$. Но это не так уж и трудно, поскольку обе стороны равенства представляют сумму всех произведений $p(e_1)\ldots p(e_n)$, взятую по всем подграфам $\{e_1,\ldots,e_n\}$ такого графа G, что $\mathrm{init}(e_i) = V_i$ и в нем существует единственный ориентированный цикл, который содержится в $\{e_1,\ldots,e_n\}$ и включает вершину V_j. Удалив любую дугу этого цикла, можно получить ориентированное дерево. Левую сторону равенства можно получить, разложив по дугам, выходящим из вершины V_j, тогда как правая сторона равенства соответствует разложению по дугам, входящим в V_j.

В некотором смысле это упражнение является комбинацией упр. 19 и 26.

28. Каждый член этого разложения имеет вид произведения $a_{1p_1}\ldots a_{mp_m}b_{1q_1}\ldots b_{nq_n}$, где $0 \le p_i \le n$ для $1 \le i \le m$ и $0 \le q_j \le m$ для $1 \le j \le n$, которое умножено на некоторый целочисленный коэффициент. Представьте это произведение в виде ориентированного графа с вершинами $\{0, u_1,\ldots,u_m, v_1,\ldots,v_n\}$ и дугами от u_i к v_{p_i} и от v_j к u_{q_j}, где $u_0 = v_0 = 0$.

Если этот диграф содержит цикл, целочисленный коэффициент будет равен нулю. Каждому такому циклу соответствует множитель вида

$$a_{i_0j_0}\,b_{j_0i_1}\,a_{i_1j_1}\,\ldots\,a_{i_{k-1}j_{k-1}}\,b_{j_{k-1}i_0}, \qquad (*)$$

где индексы $(i_0, i_1,\ldots,i_{k-1})$ различны и также различны индексы $(j_0, j_1,\ldots,j_{k-1})$. Сумма всех членов, содержащих $(*)$ в качестве множителя, является детерминантом, который получается при условии, что $a_{i_lj} \leftarrow [j = j_l]$ для $0 \le j \le n$ и $b_{j_li} \leftarrow [i = i_{(l+1) \bmod k}]$ для $0 \le i \le m$ при $0 \le l < k$, тогда как переменные в других $m + n - 2k$ строках остаются неизменными. Этот детерминант тождественно равен нулю, потому что сумма строк i_0, i_1, …, i_{k-1} в верхней части равна сумме строк j_0, j_1, …, j_{k-1} в нижней части.

С другой стороны, если ориентированный граф не содержит циклов, то целый коэффициент будет равен $+1$. Это следует из того, что все множители a_{ip_i} и b_{jq_j} должны находиться на диагонали детерминанта: если любой недиагональный элемент $a_{i_0j_0}$ выбран в строке i_0 в верхней части, то необходимо выбрать некоторый недиагональный элемент $b_{j_0i_1}$ из столбца j_0 в левой части. Следовательно, затем необходимо выбрать некоторый недиагональный элемент $a_{i_1j_1}$ из строки i_1 в верхней части и т. д., образуя замкнутый цикл.

Таким образом, коэффициент равен $+1$ тогда и только тогда, когда соответствующий диграф является ориентированным деревом с корнем 0. Количество таких членов (а значит, и количество таких ориентированных деревьев) получается за счет установки каждого a_{ij} и b_{ji} равным 1, например

$$\det\begin{pmatrix} 4 & 0 & 1 & 1 & 1\\ 0 & 4 & 1 & 1 & 1\\ 1 & 1 & 3 & 0 & 0\\ 1 & 1 & 0 & 3 & 0\\ 1 & 1 & 0 & 0 & 3 \end{pmatrix} = \det\begin{pmatrix} 4 & 0 & 1 & 1 & 1\\ -4 & 4 & 0 & 0 & 0\\ 1 & 1 & 3 & 0 & 0\\ 0 & 0 & -3 & 3 & 0\\ 0 & 0 & -3 & 0 & 3 \end{pmatrix} = \det\begin{pmatrix} 4 & 0 & 3 & 1 & 1\\ 0 & 4 & 0 & 0 & 0\\ 2 & 1 & 3 & 0 & 0\\ 0 & 0 & 0 & 3 & 0\\ 0 & 0 & 0 & 0 & 3 \end{pmatrix}$$

$$= \det\begin{pmatrix} 4 & 3\\ 2 & 3 \end{pmatrix} \cdot 4 \cdot 3 \cdot 3.$$

В общем, получим $\det\binom{n+1}{m}\binom{n}{m+1} \cdot (n+1)^{m-1} \cdot (m+1)^{n-1}$.

Замечания. Дж. Дж. Сильвестр (J. J. Sylvester) рассмотрел особый случай, когда $m = n$ и $a_{10} = a_{20} = \cdots = a_{m0} = 0$ [*Quarterly J. of Pure and Applied Math.* **1** (1857), 42–56], и справедливо предположил, что общее количество членов равно $n^n(n+1)^{n-1}$. Он также утверждает без доказательства, что количество ненулевых членов равно $(n+1)^{n-1}$, если $a_{ij} = \delta_{ij}$ соответствуют всем связным графам без циклов с вершинами $\{0, 1, \ldots, n\}$. В этом особом случае он свел детерминант к виду, рассмотренному в теореме о дереве матрицы из упр. 19, например

$$\det \begin{pmatrix} b_{10}+b_{12}+b_{13} & -b_{12} & -b_{13} \\ -b_{21} & b_{20}+b_{21}+b_{23} & -b_{23} \\ -b_{31} & -b_{32} & b_{30}+b_{31}+b_{32} \end{pmatrix}.$$

Кэли (Cayley) цитирует этот результат [*Crelle* **52** (1856), 279], приписывая его Сильвестру; таким образом, по иронии судьбы теорема о количестве таких графов часто приписывается Кэли.

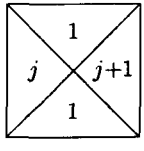
Меняя знак элементов первых m строк данного детерминанта на противоположный, а затем меняя знак элементов первых m столбцов, можно свести данное упражнение к теореме о матрице дерева.

[Рассмотренные в настоящем упражнении матрицы общего вида имеют большое значение для итеративных методов решения уравнений в частных производных. Их часто называют матрицами, обладающими свойством $\mathcal{A}$ (property $\mathcal{A}$) [см., например, Louis A. Hageman and David M. Young, *Applied Iterative Methods* (Academic Press, 1981), Chapter 9.]

РАЗДЕЛ 2.3.4.3

1. Корнем является пустая последовательность, а дуги проходят от $(x_1, \ldots, x_n)$ к $(x_1, \ldots, x_{n-1})$.

2. Возьмем один тетрадный тип и повернем его на 180°, чтобы получить другой тетрадный тип. Очевидно, что эти два тетрадных типа позволяют полностью покрыть плоскость (без дальнейших вращений), копируя фрагменты размером 2×2.

3. Рассмотрим множество тетрадных типов 

для всех положительных целых чисел j. Правую половину плоскости можно покрыть несчетным числом способов, но при размещении любого квадрата в ее центре устанавливается конечный предел расстояния, на которое это покрытие может быть продолжено влево.

4. Необходимо последовательно перебрать все возможные варианты покрытия с помощью блоков размером $n \times n$ для $n = 1, 2, \ldots$, проводя поиск тороидальных решений внутри блоков. Если покрыть плоскость нельзя, то согласно лемме о бесконечном дереве существует такое n, для которого нет решений размером $n \times n$. А если *есть* способ покрытия плоскости, то согласно этому предположению существует такое n, для которого есть решение размером $n \times n$, содержащее прямоугольник тороидального решения. Следовательно, в любом случае алгоритм прекратит работу за конечное число шагов. (Но, как будет показано в следующем упражнении, это утверждение ложно: на самом деле нет алгоритма, который позволяет за конечное число шагов определить, существует ли способ покрытия плоскости с помощью заданного множества тетрадных типов.)

5. Начнем с того, что фрагменты $\begin{smallmatrix} \alpha & \beta \\ \gamma & \delta \end{smallmatrix}$ должны присутствовать в любом решении в виде групп 2×2. Тогда необходимо выполнить следующие действия. Этап 1: рассмотрим только

α-тетрады и покажем, что фрагмент $\begin{smallmatrix} a & b \\ c & d \end{smallmatrix}$ должен повторяться в α-тетрадах в виде групп 2×2. Этапы $n > 1$: определим фрагмент, который должен находиться в крестообразной области с высотой и шириной $2^n - 1$, причем внутренняя часть крестов содержит фрагмент вида $\begin{smallmatrix} Na & Nb \\ Nc & Nd \end{smallmatrix}$, который повторяется по всей плоскости.

Например, после этапа 3 содержимое блоков 7×7 по всей плоскости будет отделяться друг от друга полосами единичной ширины через каждые восемь единиц. Блоки 7×7 с Na-тетрадой в центре имеют вид

αa	βKQ	αb	βQP	αa	βBK	αb
γPJ	δNa	γRB	δQK	γLJ	δNb	γPB
αc	βDS	αd	βQTY	αc	βBS	αd
γPQ	δPJ	γPXB	δNa	γRQ	δRB	γRB
αa	βUK	αb	βDP	αa	βBK	αb
γTJ	δNc	γSB	δDS	γST	δNd	γTB
αc	βQS	αd	βDT	αc	βBS	αd

.

"Крест" образуют средний столбец и средняя строка, которые заполняются на этапе 3; другие четыре квадрата 3×3 заполняются на этапе 2; квадраты справа и снизу от этого квадрата 7×7 являются частью креста размером 15×15, который заполняется на этапе 4.

Аналогичное построение, которое приводит к набору только из 35 тетрадных типов и содержит только тороидальные решения, можно найти в работе R. M. Robinson, *Inventiones Math.* **12** (1971), 177–209. Робинсон также продемонстрировал набор из *шести* квадратообразных форм, которые покрывают плоскость только нетороидальным образом, даже с использованием вращений и зеркальных отображений. В 1974 году Роджер Пенроуз (Roger Penrose) обнаружил набор из *двух* многоугольников, которые основываются не на квадратной решетке, а на "золотом" отношении и покрывают всю плоскость только апериодически. Это приводит к множеству всего 16 тетрадных типов лишь с нетороидальными решениями [см. B. Grünbaum and G. C. Shephard, *Tilings and Patterns* (Freeman, 1987), Chapters 10–11; Martin Gardner, *Penrose Tiles to Trapdoor Ciphers* (Freeman, 1989), Chapters 1–2].

6. Пусть k и m фиксированы. Рассмотрим ориентированное дерево, каждая вершина которого представляет для некоторого n одно из разбиений $\{1, \ldots, n\}$ на k частей, не содержащих арифметических прогрессий длины m. Узел, который представляет разбиение $\{1, \ldots, n+1\}$, является ребенком одного из разбиений $\{1, \ldots, n\}$, если оба они совпадают в части $\{1, \ldots, n\}$. Если бы существовал бесконечный путь от корня этого дерева, можно было бы разделить *все* целые числа на k множеств без арифметических прогрессий длины m. Следовательно, согласно лемме о бесконечном дереве и теореме Ван дер Вардена это дерево конечно. (Если $k = 2$ и $m = 3$, его структуру можно достаточно быстро определить вручную и наименьшим значением N будет 9.)

7. Положительные целые числа можно разбить на два таких множества S_0 и S_1, которые *не содержат* бесконечной *вычислимой* последовательности (см. упр. 3.5–32). Так, в частности, они не содержат бесконечной арифметической прогрессии. Теорема K здесь неприменима, потому что не существует способа включения частичных решений в дереве с конечными степенями в каждой вершине.

8. Назовем контрпримером последовательности бесконечную последовательность деревьев, для которой нарушается теорема Крускала, если таковые последовательности вообще существуют. Предположим, что данная теорема неверна, и в этом случае пусть T_1 — такое дерево с минимально возможным количеством узлов, что T_1 может быть первым

деревом в контрпримере последовательности. Если выбраны $T_1, \ldots, T_j$, пусть T_{j+1} — такое дерево с минимально возможным количеством узлов, что $T_1, \ldots, T_j, T_{j+1}$ является началом контрпримера последовательности. Этот процесс определяет контрпример последовательности $\langle T_n \rangle$. Ни одно из деревьев T не является только корнем. Рассмотрим теперь эту последовательность более внимательно.

(a) Предположим, что имеется последовательность $T_{n_1}, T_{n_2}, \ldots$, контрпримером которой является $l(T_{n_1}), l(T_{n_2}), \ldots$. Это невозможно, ибо последовательность $T_1, \ldots, T_{n_1-1}, l(T_{n_1}), l(T_{n_2}), \ldots$ была бы контрпримером последовательности, что противоречит определению T_{n_1}.

(b) Согласно (a) существует только конечное количество j, для которых $l(T_j)$ нельзя вложить в $l(T_k)$ для любого $k > j$. Следовательно, для n_1, большего, чем такое j, можно получить подпоследовательность, для которой $l(T_{n_1}) \subseteq l(T_{n_2}) \subseteq l(T_{n_3}) \subseteq \cdots$.

(c) Теперь согласно результату упр. 2.3.2-22 $r(T_{n_j})$ нельзя вставить в $r(T_{n_k})$ для любого $k > j$, так как в противном случае $T_{n_j} \subseteq T_{n_k}$. Значит, $T_1, \ldots, T_{n_1-1}, r(T_{n_1}), r(T_{n_2}), \ldots$ — контрпример последовательности. Но это противоречит определению T_{n_1}.

Замечания. Крускал в своей работе [*Trans. Amer. Math. Soc.* **95** (1960), 210–225] на самом деле доказал более "сильный" результат на основе более "слабого" понятия вставки. Его теорема не следует непосредственно из леммы о бесконечном дереве, хотя оба результата, в общем, подобны. Действительно, Кениг доказал особый случай теоремы Крускала о том, что не существует бесконечной последовательности парных несравнимых n-кортежей неотрицательных целых чисел, где под сравнимостью подразумевается, что все компоненты одного кортежа меньше соответствующих компонентов другого кортежа или равны им [*Matematikai és Fizikai Lapok* **39** (1932), 27–29]. Дальнейшее развитие этой интересной темы можно найти в работе *J. Combinatorial Theory* **A13** (1972), 297–305, а применение результатов для изучения вопроса о том, закончится ли алгоритм, — в работе Н. Дершовица [N. Dershowitz, *Inf. Proc. Letters* **9** (1979), 212–215].

РАЗДЕЛ 2.3.4.4

1. $\ln A(z) = \ln z + \sum_{k\ge1} a_k \ln\Big(\frac{1}{1-z^k}\Big) = \ln z + \sum_{k,t\ge1} \frac{a_k z^{kt}}{t} = \ln z + \sum_{t\ge1} \frac{A(z^t)}{t}.$

2. Дифференцируя и приравнивая коэффициенты z^n, получим тождество

$$na_{n+1} = \sum_{k\ge1} \sum_{d\backslash k} d a_d a_{n+1-k},$$

а затем изменим порядок суммирования.

4. (a) $A(z)$ сходится по крайней мере для $|z| < \frac{1}{4}$, так как a_n меньше числа *упорядоченных* деревьев b_{n-1}. Поскольку $A(1)$ — бесконечно и все коэффициенты положительны, существует такое положительное число $\alpha \le 1$, что $A(z)$ сходится для $|z| < \alpha$, и существует особая точка $z = \alpha$. Пусть $\psi(z) = A(z)/z$; так как $\psi(z) > e^{z\psi(z)}$, ясно, что из $\psi(z) = m$ следует $z < \ln m/m$, поэтому функция $\psi(z)$ ограничена и существует предел $\lim_{z\to\alpha-0} \psi(z)$. Таким образом, $\alpha < 1$ и согласно предельной теореме Абеля получим $a = \alpha\cdot\exp(a+\frac{1}{2}A(\alpha^2)+\frac{1}{3}A(\alpha^3)+\cdots)$.

(b) $A(z^2), A(z^3), \ldots$ — аналитические для $|z| < \sqrt{\alpha}$, а $\frac{1}{2}A(z^2)+\frac{1}{3}A(z^3)+\cdots$ равномерно сходится в меньшем круге.

(c) Если $\partial F/\partial w = a - 1 \ne 0$, то из теоремы о неявной функции следует, что существует такая аналитическая функция $f(z)$ в окрестности $(\alpha, a/\alpha)$, что $F(z, f(z)) = 0$. Но из этого получается $f(z) = A(z)/z$, а это противоречит тому факту, что $A(z)$ имеет особую точку $z = \alpha$.

(d) Это очевидно.

(e) $\partial F/\partial w = A(z) - 1$ и $|A(z)| < A(\alpha) = 1$, поскольку все коэффициенты $A(z)$ положительны. Следовательно, как и для (c), функция $A(z)$ регулярна во всех таких точках.

(f) Вблизи $(\alpha, 1/\alpha)$ имеем тождество $0 = \beta(z-\alpha) + (\alpha/2)(w - 1/\alpha)^2 +$ члены более высокого порядка, где $w = A(z)/z$, поэтому w — аналитическая функция от $\sqrt{z-\alpha}$ согласно теореме о неявной функции. Следовательно, существует область $|z| < \alpha_1$ с разрезом $[\alpha, \alpha_1]$, в которой $A(z)$ имеет указанный вид. (Знак "минус" выбран потому, что знак "плюс" в конце концов приводит к отрицательным коэффициентам.)

(g) Любая функция указанного вида имеет коэффициенты, асимптотически равные $\frac{\sqrt{2\beta}}{\alpha^n}\binom{1/2}{n}$. Обратите внимание, что

$$\binom{3/2}{n} = O\left(\frac{1}{n}\binom{1/2}{n}\right).$$

Более подробное описание этих приемов и асимптотические значения для количества свободных деревьев можно найти в работе R. Otter, *Ann. Math.* (2) **49** (1948), 583–599.

5. $$c_n = \sum_{j_1+2j_2+\cdots=n} \binom{c_1+j_1-1}{j_1}\cdots\binom{c_n+j_n-1}{j_n} - c_n, \quad n > 1.$$

Следовательно, получим

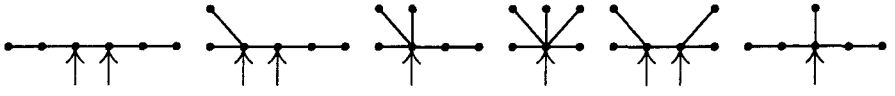
$$2C(z) + 1 - z = (1-z)^{-c_1}(1-z^2)^{-c_2}(1-z^3)^{-c_3}\ldots = \exp(C(z) + \frac{1}{2}C(z^2) + \cdots).$$

Поэтому $C(z) = z + z^2 + 2z^3 + 5z^4 + 12z^5 + 33z^6 + 90z^7 + 261z^8 + 766z^9 + \cdots$. Для $n > 1$ количество последовательно-параллельных сетей с n ребрами равно $2c_n$ [см. P. A. MacMahon, *Proc. London Math. Soc.* **22** (1891), 330–339].

6. $zG(z)^2 = 2G(z) - 2 - zG(z^2)$; $G(z) = 1 + z + z^2 + 2z^3 + 3z^4 + 6z^5 + 11z^6 + 23z^7 + 46z^8 + 98z^9 + \cdots$. Функция $F(z) = 1 - zG(z)$ удовлетворяет более простому соотношению $F(z^2) = 2z + F(z)^2$. [J. H. M. Wedderburn, *Annals of Math.* **24** (1922), 121–140.]

7. $g_n = ca^n n^{-3/2}(1 + O(1/n))$, где $c \approx 0.7916031835775$, $a \approx 2.483253536173$.

8.



9. При наличии двух центроидов, рассмотрев путь от одного центроида к другому, получим, что между ними не должно быть промежуточных вершин, поэтому два центроида должны быть смежными. Так как дерево не может содержать три взаимно смежных центроида, то их может быть не более двух.

10. Если X и Y — смежные вершины, пусть $s(X,Y)$ — количество вершин в Y-поддереве узла X. Тогда $s(X,Y)+s(Y,X) = n$. Согласно приведенным в этом разделе аргументам, если Y является центроидом, вес X равен $s(X,Y)$. Следовательно, если X и Y — центроиды, то вес узла X = весу узла $Y = n/2$.

На основе этого определения и приведенных в данном разделе аргументов можно показать, что если $s(X,Y) \geq s(Y,X)$, то существует центроид в Y-поддереве узла X. Поэтому, если два свободных дерева с m вершинами соединены ребром между узлами X и Y, получим свободное дерево, в котором $s(X,Y) = m = s(Y,X)$ и в котором должны существовать два центроида (а именно. X и Y).

[Прекрасным упражнением в программировании было бы создание программы для вычисления весов $s(X,Y)$ для всех смежных узлов X и Y за $O(n)$ шагов, так как на основе этих данных можно быстро найти узлы-центроиды. Эффективный алгоритм быстрого поиска расположения центроида был впервые предложен А. Дж. Голдманом; см. A. J. Goldman, *Transportation Sci.* **5** (1971), 212–221.]

11. $zT(z)^t = T(z) - 1$, тогда $z + T(z)^{-t} = T(z)^{1-t}$. Из соотношения 1.2.9–(21) следует, что $T(z) = \sum_n A_n(1, -t)z^n$, поэтому количество t-арных деревьев равно

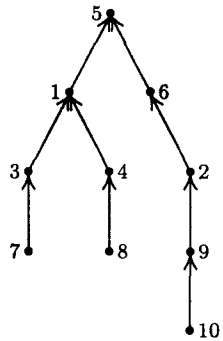
$$\binom{1+tn}{n}\frac{1}{1+tn} = \binom{tn}{n}\frac{1}{(t-1)n+1}.$$

12. Рассмотрим ориентированный граф, который имеет одну дугу от V_i к V_j для всех $i \neq j$. Матрица A_0 из упр. 2.3.4.2-19 является комбинаторной матрицей размера $(n-1) \times (n-1)$ с равными $n-1$ диагональными элементами и равными -1 недиагональными элементами. Поэтому ее детерминант равен

$$(n + (n-1)(-1))n^{n-2} = n^{n-2},$$

т. е. количеству ориентированных деревьев с заданным корнем. (Здесь можно было бы также использовать результат упр. 2.3.4.2-20.)

13.



14. Да, справедливо, поскольку этот корень не станет листом, пока не будут удалены все ветви.

15. В каноническом представлении $V_1, V_2, \ldots, V_{n-1}, f(V_{n-1})$ является топологической сортировкой ориентированного дерева, которое рассматривается как ориентированный граф. Но, вообще говоря, этот порядок может и не соответствовать порядку вывода в алгоритме 2.2.3T. Для определения значений V_1, V_2, ..., V_{n-1} алгоритм 2.2.3T может быть изменен соответствующим образом, если операцию вставки в очередь на шаге T6 заменить процедурой, которая таким образом настраивает связи, что элементы списка от начала к концу располагаются в порядке возрастания. Тогда эта очередь становится очередью по приоритету.

(Однако очередь по приоритету общего типа не нужна для поиска канонического представления. Потребуется только пройти через вершины от 1 до n, выполняя поиск листьев и в то же время отсекая все пути от новых листьев, которые меньше указателя отсечения; см. следующее упражнение.)

16. D1. Установить $\texttt{C[1]} \leftarrow \cdots \leftarrow \texttt{C}[n] \leftarrow 0$, затем установить $\texttt{C}[f(V_j)] \leftarrow \texttt{C}[f(V_j)] + 1$ для $1 \le j \le n$. (Таким образом, вершина k является листом тогда и только тогда, когда $\texttt{C}[k] = 0$.) Установить $k \leftarrow 0$ и $j \leftarrow 1$.

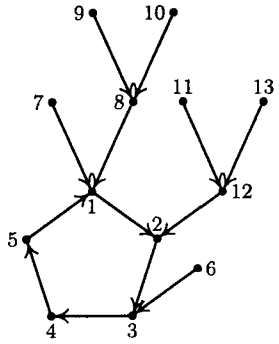
D2. Увеличивать k до тех пор, пока не будет выполнено условие $\texttt{C}[k] = 0$, а затем установить $l \leftarrow k$.

D3. Установить $\texttt{PARENT}[l] \leftarrow f(V_j)$, $l \leftarrow f(V_j)$, $\texttt{C}[l] \leftarrow \texttt{C}[l] - 1$ и $j \leftarrow j + 1$.

D4. Если $j = n$, то установить $\texttt{PARENT}[l] \leftarrow 0$ и прекратить выполнение алгоритма.

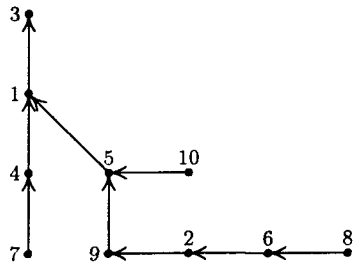
D5. Если $\texttt{C}[l] = 0$ и $l < k$, перейти к шагу D3; в противном случае вернуться к шагу D2. ∎

17. Должен существовать в точности один цикл x_1, x_2, ..., x_k, где $f(x_j) = x_{j+1}$ и $f(x_k) = x_1$. Перечислим все такие f с циклом длиной k, что итерации каждого x в конце концов будут включаться в этот цикл. Определим каноническое представление $f(V_1)$, $f(V_2)$, ..., $f(V_{m-k})$ так же, как в данном разделе. $f(V_{m-k})$ находится в цикле, поэтому продолжим создание "канонического представления", записывая остальную часть цикла $f(f(V_{m-k}))$, $f(f(f(V_{m-k})))$ и т. д. Например, функция с $m = 13$ для показанного здесь графа приводит к такому представлению: 3, 1, 8, 8, 1, 12, 12, 2, 3, 4, 5, 1. Получим последовательность $m-1$ чисел, в которой последние k чисел будут различными. И наоборот, начиная с такой последовательности, можно выполнить обратное построение (при условии, что известно k). Следовательно, существует в точности $m^k m^{m-k-1}$ таких функций, содержащих k-цикл. (Для знакомства с другими аналогичными результатами обратитесь к упр. 3.1–14. Формулу $m^{m-1}Q(m)$ впервые получил Л. Кац; он привел ее в работе L. Katz, *Annals of Math. Statistics* **26** (1955), 512–517.)



18. Для реконструкции дерева на основе последовательности s_1, s_2, ..., s_{n-1} начнем с s_1 в качестве корня и последовательно будем присоединять дуги, ведущие в s_1, s_2, ... Если вершина s_k ранее встречалась, то оставим начальную вершину дуги, ведущей к s_{k-1}, без имени, в противном случае присвоим этой вершине имя s_k. После размещения $n-1$ дуг присвоим имена всем безымянными вершинам с помощью цифр, которые еще не встречались, в порядке возрастания по мере их появления.

Например, на основе представления 3, 1, 4, 1, 5, 9, 2, 6, 5 можно построить дерево, показанное справа. Между этим методом и методом, описанным в данном разделе, не существует никакой простой связи. О других представлениях речь идет в E. H. Neville, *Proc. Cambridge Phil. Soc.* **49** (1953), 381–385.



19. Каноническое представление имеет точно $n-k$ различных величин, поэтому перечислим последовательности $n-1$ чисел, которые обладают данным свойством. Таким образом, ответ — $n^{\underline{n-k}}\left\{{n-1 \atop n-k}\right\}$.

20. Рассмотрим каноническое представление таких деревьев. Тогда возникает вопрос: сколько членов разложения $(x_1 + \cdots + x_n)^{n-1}$ имеют k_0 нулевых степеней, k_1 степеней 1 и т. д. Понятно, что каждое такое число равно коэффициенту такого члена, умноженного на количество таких членов, а именно:

$$\frac{(n-1)!}{(0!)^{k_0}(1!)^{k_1}\ldots(n!)^{k_n}} \times \frac{n!}{k_0!\,k_1!\ldots k_n!}.$$

21. Для четного количества вершин $2m$ таких деревьев не существует, а для нечетного количества вершин $n = 2m+1$ ответ можно получить на основе упр. 20 с $k_0 = m+1$, $k_2 = m$, а именно — $\binom{2m+1}{m}(2m)!/2^m$.

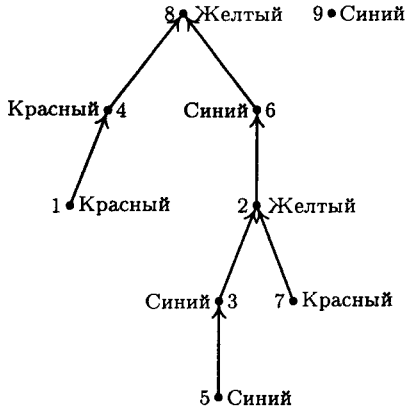
22. Точно n^{n-2}; действительно, если X — некоторая вершина, то существует взаимно однозначное соответствие между свободными деревьями и ориентированными деревьями с корнем X.

23. Каждое непомеченное упорядоченное дерево можно пометить $n!$ способами, и все такие помеченные упорядоченные деревья будут различными. Поэтому их общее количество равно $n!\,b_{n-1} = (2n-2)!/(n-1)!$.

24. Деревьев с фиксированным одним корнем столько же, сколько деревьев с другим корнем, поэтому ответ будет равен ответу из упр. 23, умноженному на $1/n$. Например, в данном случае ответ — 30.

25. $r(n,q) = (n-q)n^{q-1}$ для $0 \le q < n$. (Особый случай — $s = 1$ для уравнения (24).)

26. ($k = 7$)



27. Пусть дана такая функция g с областью определения $\{1, 2, \ldots, r\}$ и областью значений $\{1, 2, \ldots, q\}$, что добавление дуг от V_k к $U_{g(k)}$ не приводит к появлению ориентированных циклов. Построим последовательность $a_1, \ldots, a_r$. Назовем вершину V_k "свободной", если не существует ориентированных путей от V_j к V_k для любых $j \ne k$. Поскольку не существует ориентированных циклов, должна существовать по крайней мере одна свободная вершина. Пусть b_1 — наименьшее целое число, для которого вершина V_{b_1} является свободной. Предположив, что выбраны числа $b_1, \ldots, b_t$, допустим, что b_{t+1} — наименьшее целое число, отличное от $b_1, \ldots, b_t$, для которых $V_{b_{t+1}}$ является свободной вершиной в графе, полученном путем удаления дуг от V_{b_k} к $U_{g(b_k)}$ для $1 \le k \le t$ Это правило определяет перестановку $b_1 b_2 \ldots b_r$ целых чисел $\{1, 2, \ldots, r\}$. Пусть $a_k = g(b_k)$ для $1 \le k \le r$, следовательно, определена такая последовательность, что $1 \le a_k \le q$ для $1 \le k < r$ и $1 \le a_r \le p$.

И наоборот, если дана такая последовательность $a_1, \ldots, a_r$, назовем вершину V_k свободной, если не существует такого j, для которого $a_j > p$ и $f(a_j) = k$. Поскольку $a_r \le p$, имеется не более $r-1$ несвободных вершин. Пусть b_1 — наименьшее целое, для которого вершина V_{b_1} является свободной. Предположив, что выбраны $b_1, \ldots, b_t$, допустим, что b_{t+1} — наименьшее целое, отличное от $b_1, \ldots, b_t$, для которых вершина $V_{b_{t+1}}$ является свободной по отношению к последовательности $a_{t+1}, \ldots, a_r$. Данное правило определяет перестановку $b_1 b_2 \ldots b_r$ целых чисел $\{1, 2, \ldots, r\}$. Пусть $g(b_k) = a_k$ для $1 \le k \le r$; это определяет такую функцию, что добавление дуг от V_k к $U_{g(k)}$ не приводит к появлению ориентированных циклов.

28. Пусть f — любая из n^{m-1} функций с областью определения $\{2, \ldots, m\}$ и областью значений $\{1, 2, \ldots, n\}$. Рассмотрим ориентированный граф с вершинами $U_1, \ldots, U_m, V_1, \ldots, V_n$ и дугами от U_k к $V_{f(k)}$ для $1 < k \le m$. Применим результат упр. 27 с $p = 1$, $q = m$, $r = n$, чтобы показать, что существует m^{n-1} способов добавления дополнительных дуг от V к U для получения ориентированного дерева с корнем U_1. Поскольку между рассматриваемым множеством свободных деревьев и множеством ориентированных деревьев

с корнем U_1 существует взаимно однозначное соответствие, ответ будет таким: $n^{m-1}m^{n-1}$. [Это построение можно значительно обобщить; см. D. E. Knuth, *Canadian J. Math.* **20** (1968), 1077–1086.]

29. Если $y = x^t$, то $(tz)y = \ln y$ и достаточно доказать тождество для $t = 1$. Затем, если $zx = \ln x$, на основании результата упр. 25 получим, что $x^m = \sum_k E_k(m,1)z^k$ для неотрицательных целых чисел m. Следовательно,

$$x^r = e^{zxr} = \sum_k \frac{(zxr)^k}{k!} = \sum_{j,k} \frac{r^k z^{k+j} E_j(k,1)}{k!} = \sum_k \frac{z^k}{k!} \sum_j \binom{k}{j} j! E_j(k-j,1) r^{k-j}$$

$$= \sum_k \frac{z^k}{k!} \binom{k-1}{j} k^j r^{k-j} = \sum_k z^k E_k(r,1).$$

[В упр. 4.7–22 приводится значительно более общий результат.]

30. Каждый описанный граф определяет множество $C_x \subseteq \{1,\dots,n\}$, где j принадлежит C_x тогда и только тогда, когда существует путь от t_j к r_i для некоторого $i \le x$. Для данного множества C_x каждый описанный граф состоит из двух независимых частей: одна часть принадлежит множеству $x(x+\epsilon_1 z_1+\cdots+\epsilon_n z_n)^{\epsilon_1+\cdots+\epsilon_n-1}$ графов с вершинами r_i, s_{jk}, t_j для $i \le x$ и $j \in C_x$, где $\epsilon_j = [j \in C_x]$, а другая — множеству $y(y+(1-\epsilon_1)z_1+\cdots+(1-\epsilon_n)z_n)^{(1-\epsilon_1)+\cdots+(1-\epsilon_n)-1}$ графов с остальными вершинами.

31. $G(z) = z + G(z)^2 + G(z)^3 + G(z)^4 + \cdots = z + G(z)^2/(1-G(z))$. Следовательно, $G(z) = \frac{1}{4}(1+z-\sqrt{1-6z+z^2}) = z + z^2 + 3z^3 + 11z^4 + 45z^5 + \cdots$. [*Замечания.* Другая задача, эквивалентная данной, была поставлена и решена Ф. В. К. Э. Шредером [F. W. K. E Schröder, *Zeitschrift für Mathematik und Physik* **15** (1870), 361–376], который определил количество способов вставки непересекающихся диагоналей в выпуклом $(n+1)$-угольнике. Эти числа для $n > 1$ всего вдвое меньше чисел, полученных в упр. 2.2.1–11, так как согласно грамматике Пратта в ассоциированном дереве допускается наличие корня со степенью 1. Асимптотическое значение вычисляется в упр. 2.2 1–12. Любопытно, что значение $[z^{10}]\,G(z) = 103049$, похоже, было найдено еще Гиппархом во 2 веке до н. э. как количество "утвердительных сложных суждений, которые можно вывести на основании только десяти простых суждений"; см. R. P. Stanley, *AMM* **104** (1997), 344–350.]

32. Ни одного, если $n_0 \ne 1 + n_2 + 2n_3 + 3n_4 + \cdots$ (см. упр. 2.3–21). В противном случае

$$(n_0 + n_1 + \cdots + n_m - 1)!/n_0!\,n_1!\dots n_m!.$$

Для доказательства этого результата напомним, что непомеченное дерево с $n = n_0 + n_1 + \cdots + n_m$ узлами характеризуется последовательностью $d_1\,d_2\dots d_n$ степеней узлов в обратном порядке обхода (раздел 2.3.3). Более того, данная последовательность степеней соответствует дереву тогда и только тогда, когда $\sum_{j=1}^{k}(1-d_j) > 0$ для $0 < k \le n$. (Это важное свойство польской системы обозначений (или польской нотации) легко доказывается с помощью метода индукции; см. алгоритм 2.3.3F с функцией построения дерева f, которая подобна функции TREE из раздела 2.3.2.) В частности, d_1 должно быть равно 0. Следовательно, ответом для этой задачи будет количество последовательностей $d_2\dots d_n$, в которых n_j раз встречается j для $j > 0$, а именно — полиномиальный коэффициент

$$\binom{n-1}{n_0-1,\,n_1,\dots,n_m}$$

минус количество таких последовательностей $d_2\dots d_n$, для которых $\sum_{j=2}^{k}(1-d_j) < 0$ для *определенного* $k \ge 2$.

Эти последовательности можно перечислить следующим образом. Пусть t — такое минимальное значение, что $\sum_{j=2}^{t}(1-d_j) < 0$. Тогда $\sum_{j=2}^{t}(1-d_j) = -s$, где $1 \le s < d_t$,

и можно построить подпоследовательность $d'_2 \dots d'_n = d_{t-1} \dots d_2 0 d_{t+1} \dots d_n$, в которой n_j раз встречается j для $j \neq d_t$, $n_j - 1$ раз встречается j для $j = d_t$. Тогда $\sum_{j=2}^{k}(1 - d'_j)$ равна d_t для $k = n$ и эта сумма равна $d_t - s$ для $k = t$. Для $k < t$ получим

$$\sum_{2\le j<t}(1-d_j) - \sum_{2\le j\le t-k}(1-d_j) \le \sum_{2\le j<t}(1-d_j) = d_t - s - 1.$$

Отсюда следует, что для данного s и любой последовательности $d'_2 \dots d'_n$ это построение можно обратить. Значит, количество последовательностей $d_2 \dots d_n$ для заданных значений d_t и s можно выразить с помощью полиномиального коэффициента

$$\binom{n-1}{n_0, \dots, n_{d_t} - 1, \dots, n_m}.$$

Тогда количество последовательностей $d_2 \dots d_n$, которые соответствуют деревьям, можно получить за счет суммирования всех допустимых значений d_t и s:

$$\sum_{j=0}^{m}(1-j)\binom{n-1}{n_0, \dots, n_j - 1, \dots, n_m} = \frac{(n-1)!}{n_0!\, n_1! \dots n_m!}\sum_{j=0}^{m}(1-j)n_j,$$

причем последняя сумма равна 1.

Еще более простое доказательство этого результата предложено Дж. Н. Рэйни (G. N. Raney) [см. *Transactions of the American Math. Society* **94** (1960), 441–451]. Если $d_1\, d_2 \dots d_n$ — произвольная последовательность, в которой n_j раз встречается j, то существует в точности одна циклическая перестановка $d_k \dots d_n d_1 \dots d_{k-1}$, которая соответствует дереву, а именно: перестановка, в которой k — такое максимальное значение, что сумма $\sum_{j=1}^{k}(1-d_j)$ является минимальной. [По-видимому, этот результат для бинарных деревьев впервые был получен Ч. С. С. Пирсом (C. S. S. Peirce) в неопубликованной рукописи; см. его книгу *New Elements of Mathematics* **4** (The Hague: Mouton, 1976), 303–304. Для t-арных деревьев доказательство предложено Дворецким и Моцкиным; см. *Duke Math. J.* **14** (1947), 305–313.]

Еще одно доказательство предложено Дж. М. Бергманом (G. M. Bergman), в котором согласно методу индукции $d_k d_{k+1}$ заменяется на $(d_k + d_{k+1} - 1)$, если $d_k > 0$ [*Algebra Universalis* **8** (1978), 129–130].

Описанные выше методы можно обобщить и показать, что количество упорядоченных и непомеченных лесов с f деревьями и n_j узлами степени j равно $(n-1)!\, f / n_0!\, n_1! \dots n_m!$, если выполняется условие $n_0 = f + n_2 + 2n_3 + \cdots$.

33. Рассмотрим количество деревьев с n_1 узлами, которые помечены номером 1, с n_2 узлами, которые помечены номером 2, и т. д., таких, что каждый узел с меткой (номером) j имеет степень e_j. Пусть это число равно $c(n_1, n_2, \dots)$ с фиксированными степенями e_1, e_2, ... Производящая функция $G(z_1, z_2, \dots) = \sum c(n_1, n_2, \dots) z_1^{n_1} z_2^{n_2} \dots$ удовлетворяет тождеству $G = z_1 G^{e_1} + \cdots + z_r G^{e_r}$, так как $z_j G^{e_j}$ перечисляет деревья с корнями, помеченными номером j. Согласно результату предыдущего упражнения

$$c(n_1, n_2, \dots) = \begin{cases} \dfrac{(n_1 + n_2 + \cdots - 1)!}{n_1!\, n_2! \dots}, & \text{если } (1-e_1)n_1 + (1-e_2)n_2 + \cdots = 1; \\ 0 & \text{в остальных случаях.} \end{cases}$$

Вообще говоря, поскольку G^f перечисляет все упорядоченные леса с такими ярлыками, для целых $f > 0$ получим

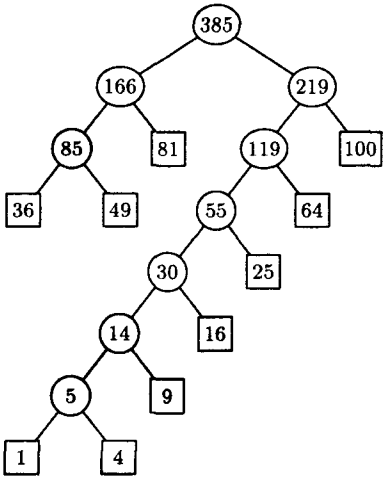
$$w^f = \sum_{f=(1-e_1)n_1+(1-e_2)n_2+\cdots} \frac{(n_1 + n_2 + \cdots - 1)!\, f}{n_1!\, n_2! \dots} z_1^{n_1} z_2^{n_2} \dots.$$

Эти формулы имеют смысл, когда $r = \infty$, и, по существу, они эквивалентны формуле обратного преобразования Лагранжа.

РАЗДЕЛ 2.3.4.5

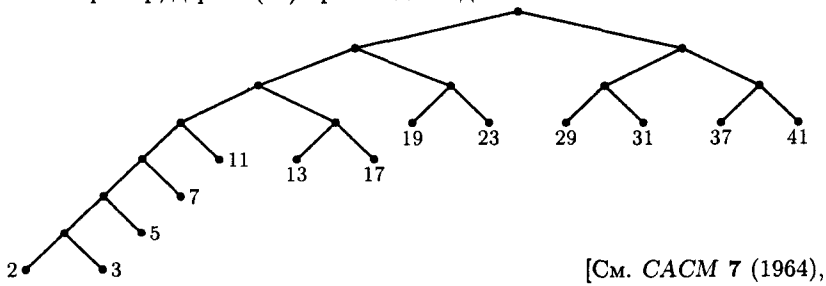
1. Всего существует $\binom{8}{5}$ таких деревьев, поскольку узлы с номерами 8–12 можно присоединить в любом из 8 мест под узлами 4–7.

2.



3. Согласно методу индукции по m данное условие является необходимым. И наоборот, если $\sum_{j=1}^{m} 2^{-l_j} = 1$, нужно построить расширенное бинарное дерево с длинами пути $l_1, \ldots, l_m$. Если $m = 1$, имеем $l_1 = 0$, и построение становится тривиальным. В противном случае можно предположить, что l упорядочены так, что $l_1 = l_2 = \cdots = l_q > l_{q+1} \geq l_{q+2} \geq \cdots \geq l_m > 0$ для некоторого q с $1 \leq q \leq m$. Тогда $2^{l_1-1} = \sum_{j=1}^{m} 2^{l_1-l_j-1} = \frac{1}{2}q +$ целое число. Следовательно q — четно. На основании метода индукции по m можно доказать, что существует дерево с длинами пути $l_1 - 1, l_3, l_4, \ldots, l_m$. В таком дереве заменим один из внешних узлов на уровне $l_1 - 1$ внутренним узлом, дети которого находятся на уровне $l_1 = l_2$.

4. Сначала построим дерево по методу Хаффмэна. Если $w_j < w_{j+1}$, то $l_j \geq l_{j+1}$, так как это дерево является оптимальным. Построение из ответа к упр. 3 теперь позволяет получить другое дерево с теми же длинами пути и весами в соответствующей последовательности. Например, дерево (11) принимает вид



[См. *CACM* **7** (1964), 166–169.]

5. (a) $b_{np} = \sum_{\substack{k+l=n-1 \\ r+s+n-1=p}} b_{kr} b_{ls}$. Следовательно, $zB(w, wz)^2 = B(w, z) - 1$.

(b) Возьмем частную производную по w:

$$2zB(w,wz)(B_w(w,wz)+zB_z(w,wz))=B_w(w,z).$$

Значит, если $H(z)=B_w(1,z)=\sum_n h_n z^n$, найдем $H(z)=2zB(z)(H(z)+zB'(z))$. Из известной формулы для $B(z)$

$$H(z)=\frac{1}{1-4z}-\frac{1}{z}\left(\frac{1-z}{\sqrt{1-4z}}-1\right),\qquad\text{следовательно,}\qquad h_n=4^n-\frac{3n+1}{n+1}\binom{2n}{n}.$$

Среднее значение равно h_n/b_n.

(c) Асимптотически это выражение равно $n\sqrt{\pi n}-3n+O(\sqrt{n})$.

[См. решения аналогичных задач в работах John Riordan, *IBM J. Res. and Devel.* **4** (1960), 473–478; A. Rényi and G. Szekeres, *J. Australian Math. Soc.* **7** (1967), 497–507; John Riordan and N. J. A. Sloane, *J. Australian Math. Soc.* **10** (1969), 278–282; а также в упр. 2.3.1–11.]

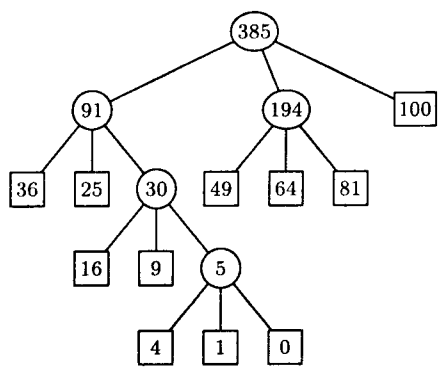
6. $n+s-1=tn$.

7. $E=(t-1)I+tn$.

8. Суммируя по частям, получим $\sum_{k=1}^n\lfloor\log_t((t-1)k)\rfloor=nq-\sum k$, где суммирование справа выполняется по таким k, что $0\le k\le n$ и $(t-1)k+1=t^j$ для некоторого j. Последнюю сумму можно представить в виде $\sum_{j=1}^q(t^j-1)/(t-1)$.

9. Воспользуйтесь методом индукции по размеру дерева.

10. Добавив (если это необходимо) дополнительные *нулевые* веса, можно предположить, что $m \bmod (t-1)=1$. Чтобы получить t-арное дерево с минимальной взвешенной длиной пути, на каждом этапе объединим наименьшие значения t и заменим их суммой этих же значений. Доказательство в таком случае выполняется, как для бинарного дерева. Искомое тернарное дерево показано справа.



Ф. К. Хван (F. K. Hwang) заметил [*SIAM J. Appl. Math.* **37** (1979), 124–127], что аналогичная процедура справедлива для построения деревьев с минимальным взвешенным путем, имеющих произвольно заданное мультимножество степеней. Для этого следует объединить t весов на каждом этапе, где t является минимально возможным.

11. Десятичная система обозначений Дьюи является двоичным представлением номеров узлов.

12. Согласно результату упр. 9 средний размер поддерева с корнем в этом узле равен внутренней длине пути, деленной на n плюс 1. (Этот результат верен как для деревьев общего типа, так и для бинарных деревьев.)

13. [См. J. van Leeuwen, *Proc. 3rd International Colloq. Automata, Languages and Programming* (Edinburgh University Press, 1976), 382–410.]

H1. [Инициализация.] Установить $A[m-1+i]\leftarrow w_i$ для $1\le i\le m$. Затем установить $A[2m]\leftarrow\infty$, $x\leftarrow m$, $i\leftarrow m+1$, $j\leftarrow m-1$, $k\leftarrow m$. (В данном алгоритме $A[i]\le\cdots\le A[2m-1]$ — очередь неиспользованных внешних весов; $A[k]\ge\cdots\ge A[j]$ — очередь неиспользованных внутренних весов, которая пуста в случае, когда $j<k$; x и y — текущие левый и правый указатели.)

H2. [Поиск правого указателя.] Если $j<k$ или $A[i]\le A[j]$, то установить $y\leftarrow i$ и $i\leftarrow i+1$; в противном случае установить $y\leftarrow j$ и $j\leftarrow j-1$.

H3. [Создание внутреннего узла.] Установить $k \leftarrow k-1$, $L[k] \leftarrow x$, $R[k] \leftarrow y$, $A[k] \leftarrow A[x] + A[y]$.

H4. [Готово?] Прекратить выполнение алгоритма, если $k = 1$.

H5. [Поиск левого указателя.] (В этом месте $j \geq k$ и очереди содержат всего k неиспользованных весов. Если $A[y] < 0$, получим $j = k$, $i = y+1$ и $A[i] > A[j]$.) Если $A[i] \leq A[j]$, то установить $x \leftarrow i$ и $i \leftarrow i+1$; в противном случае установить $x \leftarrow j$ и $j \leftarrow j-1$. Вернуться к шагу H2. ▮

14. С небольшими изменениями здесь можно использовать то же доказательство, в котором $k = m-1$. [См. *SIAM J. Appl. Math.* **21** (1971), 518.]

15. Вместо правила объединения весов $w_1 + w_2$ из (9) используем здесь такие функции объединения весов: (a) $1 + \max(w_1, w_2)$ и (b) $xw_1 + xw_2$ соответственно. Случай (a) исследован в работе M. C. Golumbic, *IEEE Trans.* **C-25** (1976), 1164–1167, а случай (b) — в работе T. C. Hu, D. Kleitman, and J. K. Tamaki, *SIAM J. Appl. Math.* **37** (1979), 246–256. Задача Хаффмэна — это предельный случай (b) при $x \to 1$, так как $\sum(1+\epsilon)^{l_j} w_j = \sum w_j + \epsilon \sum w_j l_j + O(\epsilon^2)$.

Д. Стот Паркер показал, что алгоритм, подобный алгоритму Хаффмэна, позволяет также найти минимальное значение $w_1 x^{l_1} + \cdots + w_m x^{l_m}$, когда $0 < x < 1$, если два *максимальных* веса объединяются на каждом этапе, как в случае (b). В частности, минимальное значение $w_1 2^{-l_1} + \cdots + w_m 2^{-l_m}$ для $w_1 \leq \cdots \leq w_m$ равно $w_1/2 + \cdots + w_{m-1}/2^{m-1} + w_m/2^{m-1}$. Продолжение обсуждения этой темы можно найти в работе D. E. Knuth, *J. Comb. Theory* **A32** (1982), 216–224.

16. Пусть $l_{m+1} = l'_{m+1} = 0$. Тогда

$$\sum_{j=1}^{m} w_j l_j \leq \sum_{j=1}^{m} w_j l'_j = \sum_{k=1}^{m} (l'_k - l'_{k+1}) \sum_{j=1}^{k} w_j \leq \sum_{k=1}^{m} (l'_k - l'_{k+1}) \sum_{j=1}^{k} w'_j = \sum_{j=1}^{m} w'_j l'_j ,$$

поскольку $l'_j \geq l'_{j+1}$, как и в упр. 4. Такое же доказательство можно использовать для многих других типов оптимальных деревьев, включая дерево из упр. 10.

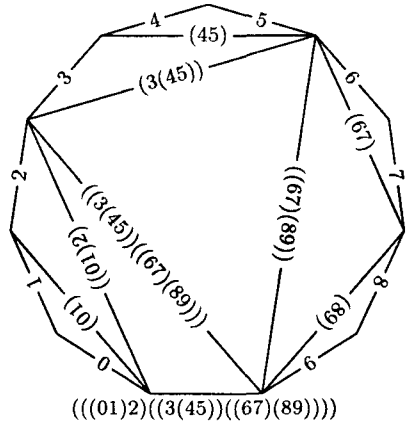
17. (a) То же, что и в упр. 14. (b) Функцию $f(n)$ можно расширить до вогнутой функции $f(x)$, а потому указанное неравенство справедливо. Тогда $F(m)$ является минимумом суммы $\sum_{j=1}^{m-1} f(s_j)$, где s_j — веса внутренних узлов расширенного бинарного дерева с весами $1, 1, \ldots, 1$. Алгоритм Хаффмэна, который позволяет в данном случае построить полное бинарное дерево с $m-1$ внутренними узлами, приводит к построению оптимального дерева. Выбор значения $k = 2^{\lceil \lg(n/3) \rceil}$ определяет бинарное дерево с теми же весами внутренних узлов, поэтому для него рекуррентное соотношение достигает минимального значения для всех n. [*SIAM J. Appl. Math.* **31** (1976), 368–378.] Значение $F(n)$ можно оценить за $O(\log n)$ шагов (см. упр. 5.2.3–20 и 5.2.3–21). Если функция $f(n)$ выпуклая, а не вогнутая (и потому $\Delta^2 f(n) \geq 0$), то решение для этой рекуррентной формулы будет получено для $k = \lfloor n/2 \rfloor$.

РАЗДЕЛ 2.3.4.6

1. Выберем одну сторону многоугольника и назовем ее базой. Для данной триангуляции (т. е. рассечения многоугольника на треугольники) пусть треугольник базы соответствует *корню бинарного дерева*, а *две другие его стороны определяют базы левого и правого* подмногоугольников, которые таким же образом соответствуют левому и правому поддеревьям. Будем выполнять эти действия рекурсивно до тех пор, пока не достигнем двусторонних многоугольников, которые соответствуют пустым бинарным деревьям.

Сформулировав соответствие иначе, пометим небазовые ребра триангулированного многоугольника целыми числами $0, \ldots, n$; когда две смежные стороны треугольника будут

помечены по часовой стрелке символами α и β, пометим третью сторону как $(\alpha\beta)$. Тогда метка базы характеризует бинарное дерево и триангуляцию. Например, многоугольник



соответствует бинарному дереву 2 3.1–(1).

2. (a) Возьмем базовую сторону, которая описана в упр 1, и присвоим ей d последователей, если эта сторона принадлежит $(d+1)$-угольнику в рассеченном диагоналями r-угольнике. Тогда другие d сторон являются базами поддеревьев. Таким образом определено соответствие между задачей Киркмана и всеми упорядоченными деревьями с $r-1$ узлами-листьями, $k+1$ узлами-не листьями и без узлов степени 1. (При $k=r-3$ получим ту же ситуацию, что и в упр. 1)

(b) Существует $\binom{r+k}{k+1}\binom{r-3}{k}$ таких последовательностей $d_1 d_2 \dots d_{r+k}$ неотрицательных целых чисел, что $r-1$ чисел из d равны 0, ни одно из них не равно 1, а сумма равна $r+k-1$. В точности одна циклическая перестановка из перестановок $d_1 d_2 \dots d_{r+k}$, $d_2 \dots d_{r+k} d_1$, $\dots$, $d_{r+k} d_1 \dots d_{r+k-1}$ удовлетворяет дополнительному свойству: $\sum_{j=1}^{q}(1-d_j) > 0$ для $1 \le q \le r+k$.

[Киркман предложил доказательство своей гипотезы в *Philos. Trans.* **147** (1857), 217–272, §22, а Кэли доказал ее без упоминания какой-либо связи с деревьями в *Proc. London Math. Soc.* **22** (1891), 237–262]

3. (a) Пусть вершины обозначены числами $\{1, 2, \dots, n\}$. Проведем связи RLINK от i к j, если i и j являются последовательно расположенными элементами одной части и $i < j$; проведем связи LLINK от j к $j+1$, если $j+1$ является наименьшим значением в его части. Тогда существует $k-1$ ненулевых связей LLINK, $n-k$ ненулевых связей RLINK и бинарное дерево с узлами $12\dots n$ при обходе в прямом порядке. Используя естественное соответствие из раздела 2.3.2, получим, что это правило определяет взаимно однозначное соответствие между "разбиениями вершин n-угольника на k непересекающихся частей" и "лесами с n вершинами и $n-k+1$ листьями". Поменяв местами связи LLINK и RLINK, можно также получить "леса с n вершинами и k листьями".

(b) Лес с n вершинами и k листьями соответствует последовательности вложенных скобок, которые содержат n левых скобок, n правых скобок и k пар скобок "()". Такие последовательности можно перечислить следующим образом.

Будем считать, что строка нулей и единиц является (m,n,k)-строкой, если в ней содержится m нулей, n единиц, а также k пар "01". Тогда 0010101001110 является $(7,6,4)$-строкой. Всего существует $\binom{m}{k}\binom{n}{k}$ таких (m,n,k)-строк, поскольку любые нули и единицы могут образовывать пары 01

Пусть $S(\alpha)$ — число нулей в α минус число единиц. Назовем строку σ *хорошей*, если $S(\alpha) \ge 0$, когда α находится в префиксной части σ (иначе говоря, если из $\sigma = \alpha\beta$ следует,

что $S(\alpha) \geq 0$); в противном случае назовем строку σ *плохой*. Тогда приведенный ниже вариант "принципа отражения" из упр. 2.2.1–4 задает взаимно однозначное соответствие между плохими (n, n, k)-строками и произвольными $(n-1, n+1, k)$-строками.

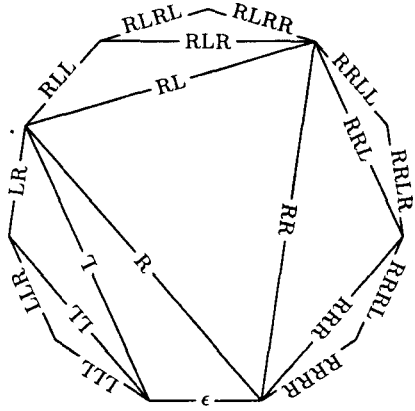
Всякую плохую (n, n, k)-строку σ можно единственным образом представить в виде $\sigma = \alpha 0 \beta$, где $\overline{\alpha}^R$ и β — хорошие строки. (Здесь $\overline{\alpha}^R$ — строка, полученная из строки α за счет ее обращения и дополнения всех ее битов.) Тогда $\sigma' = \alpha 1 \beta$ будет соответствовать $(n-1, n+1, k)$-строке. И наоборот, каждую $(n-1, n+1, k)$-строку можно единственным образом представить в виде $\alpha 1 \beta$, где $\overline{\alpha}^R$ и β — хорошие строки, а $\alpha 0 \beta$ — плохая (n, n, k)-строка.

Следовательно, количество лесов с n вершинами и k листьями равно $\binom{n}{k}\binom{n}{k} - \binom{n-1}{k}\binom{n+1}{k} = n!\,(n-1)!/(n-k+1)!\,(n-k)!\,k!\,(k-1)!$.

Замечания. Дж. Креверас (G. Kreweras) в работе *Discrete Math.* **1** (1972), 333–350, перечислил непересекающиеся разбиения другим способом. Частичное упорядочение разбиений за счет более мелкого дробления приводит к интересному частичному упорядочению лесов, которое отличается от упорядочения, обсуждаемого в упр. 2.3.3–19 [см. Y. Poupard, *Cahiers du Bureau Univ. de Recherche Opérationnelle* **16** (1971), Chapter 8; *Discrete Math.* **2** (1972), 279–288; P. Edelman, *Discrete Math.* **31** (1980), 171–180, **40** (1982), 171–179].

Третий способ определения естественного решеточного упорядочения лесов предложен Р. П. Стэнли (R. Stanley) в работе *Fibonacci Quarterly* **13** (1975), 215–232. Представим лес в виде строки σ из нулей и единиц, которые представляют упомянутые выше левые и правые скобки. В таком случае $\sigma \leq \sigma'$ тогда и только тогда, когда $S(\sigma_k) \leq S(\sigma'_k)$ для всех k, где σ_k обозначает первые k бит σ. Решетка Стэнли *дистрибутивна* в отличие от двух других.

4. Пусть $m = n+2$. Используя результат упр. 1, следует установить соответствие между триангулированными m-угольниками и $(m-1)$-строчными бордюрами. Сначала более подробно проанализируем предложенное выше соответствие, присваивая ребрам триангуляции ярлыки "верх-низ" вместо рассмотренных ярлыков "низ-вверх" так, как описывается ниже. Присвоим пустой ярлык ϵ базе, а затем рекурсивно присвоим ярлыки αL и αR противоположным ребрам треугольника, база которого помечена ярлыком α. Например, так будет выглядеть предыдущая схема при использовании новых обозначений.



Если базовое ребро в этом примере имеет ярлык 10, а другие ребра, как и прежде, имеют ярлыки от 0 до 9, можно записать $0 = 10LLL$, $1 = 10LLR$, $2 = 10LR$, $3 = 10RLL$ и т. д. В качестве базы может быть выбрано любое другое ребро; таким образом, если выбрано ребро 0, получим $1 = 0L$, $2 = 0RL$, $3 = 0RRLLL$ и т. д. Нетрудно проверить, что, если $u = v\alpha$, получим $v = u\alpha^T$, где α^T получено при чтении строки α справа налево и замене

L на R. Например, $10 = 0RRR = 1LRR = 2LR = 3RRL$ и т. д. Если u, v и w — ребра многоугольника с $w = u\alpha L\gamma$ и $w = v\beta R\gamma$, то получим $u = v\beta L\alpha^T$ и $v = u\alpha R\beta^T$.

Для данной триангуляции многоугольника, стороны которого пронумерованы числами $0, 1, \ldots, m-1$, определим (u, v) для любой пары u и v следующим образом. Пусть $u = v\alpha$, а α представим в виде матрицы размера 2×2, предполагая, что $L = \left(\begin{smallmatrix} 1 & 1 \\ 0 & 1 \end{smallmatrix}\right)$ и $R = \left(\begin{smallmatrix} 1 & 0 \\ 1 & 1 \end{smallmatrix}\right)$. Тогда (u, v) — это элемент в правом верхнем углу α. Обратите внимание на то, что матрица α^T — это транспонированная матрица α, так как $R = L^T$. Следовательно, получим $(v, u) = (u, v)$. Также необходимо отметить, что $(u, v) = 1$ тогда и только тогда, когда u_- и v_- объединены ребром триангуляции, где u_- обозначает вершину между ребрами u и $u-1$.

Пусть $(u, u) = 0$ для всех сторон многоугольника u. Можно доказать, что из $v = u\alpha$ следует

$$\alpha = \begin{pmatrix} (u, v) & (u, v+1) \\ (u+1, v) & (u+1, v+1) \end{pmatrix} \qquad \text{для всех } u \neq v, \tag{*}$$

где $u+1$ и $v+1$ — последователи u и v в направлении обхода по часовой стрелке. Доказательство этого утверждения выполняется с помощью метода индукции по m. Соотношение (*) тривиально для $m = 2$, так как два параллельных ребра u и v связаны соотношениями $u = v\epsilon$, а $\alpha = \epsilon$ — единичная матрица. Если произвольную триангуляцию дополнить некоторым ребром v с треугольником $vv'v''$, то из $v = u\alpha$ будет следовать, что $v' = u\alpha L$ и $v'' = u\alpha R$. Значит, (u, v') и (u, v'') в этом расширенном многоугольнике соответственно равны (u, v) и $(u, v) + (u, v+1)$ из исходного многоугольника. Отсюда следует, что

$$\alpha L = \begin{pmatrix} (u, v') & (u, v'') \\ (u+1, v') & (u+1, v'') \end{pmatrix} \qquad \text{и} \qquad \alpha R = \begin{pmatrix} (u, v'') & (u, v''+1) \\ (u+1, v'') & (u+1, v''+1) \end{pmatrix},$$

а (*) выполняется и для расширенного многоугольника.

Узор бордюра, который соответствует данной триангуляции, теперь определяется периодической последовательностью

$$\begin{array}{cccccccccc}
(0,1) & (1,2) & (2,3) & \ldots & (m-1,0) & (0,1) & (1,2) & \ldots \\
(0,2) & (1,3) & (2,4) & \ldots & (m-1,1) & (0,2) & (1,3) & \ldots \\
(m-1,2) & (0,3) & (1,4) & \ldots & (m-2,1) & (m-1,2) & (0,3) & \ldots \\
(m-1,3) & (0,4) & (1,5) & \ldots & (m-2,2) & (m-1,3) & (0,4) & \ldots
\end{array}$$

и т. д , пока не будут определены $m-1$ строк; последняя строка начинается с $(\lceil m/2\rceil + 1, \lceil m/2\rceil)$ для $m > 3$. С помощью условия (*) можно доказать, что этот узор является бордюром, т. е что

$$(u, v)(u+1, v+1) - (u, v+1)(u+1, v) = 1, \tag{**}$$

поскольку из $\det L = \det R = 1$ следует, что $\det\alpha = 1$. Для рассматриваемого здесь примера триангуляции получим следующее.

$$\begin{array}{cccccccccccccccccccccc}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & \ldots \\
1 & 2 & 4 & 2 & 1 & 5 & 1 & 3 & 1 & 4 & 3 & 1 & 2 & 4 & 2 & 1 & 5 & 1 & 3 & 1 & 4 & \ldots \\
2 & 1 & 7 & 7 & 1 & 4 & 4 & 2 & 2 & 3 & 11 & 2 & 1 & 7 & 7 & 1 & 4 & 4 & 2 & 2 & 3 & \ldots \\
1 & 3 & 12 & 3 & 3 & 3 & 7 & 1 & 5 & 8 & 7 & 1 & 3 & 12 & 3 & 3 & 3 & 7 & 1 & 5 & 8 & \ldots \\
3 & 2 & 5 & 5 & 8 & 2 & 5 & 3 & 2 & 13 & 5 & 3 & 2 & 5 & 5 & 8 & 2 & 5 & 3 & 2 & 13 & \ldots \\
5 & 3 & 2 & 13 & 5 & 3 & 2 & 5 & 5 & 8 & 2 & 5 & 3 & 2 & 13 & 5 & 3 & 2 & 5 & 5 & 8 & \ldots \\
3 & 7 & 1 & 5 & 8 & 7 & 1 & 3 & 12 & 3 & 3 & 3 & 7 & 1 & 5 & 8 & 7 & 1 & 3 & 12 & 3 & \ldots \\
4 & 2 & 2 & 3 & 11 & 2 & 1 & 7 & 7 & 1 & 4 & 4 & 2 & 2 & 3 & 11 & 2 & 1 & 7 & 7 & 1 & \ldots \\
5 & 1 & 3 & 1 & 4 & 3 & 1 & 2 & 4 & 2 & 1 & 5 & 1 & 3 & 1 & 4 & 3 & 1 & 2 & 4 & 2 & \ldots \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & \ldots
\end{array}$$

Отношение $(u, v) = 1$ определяет ребра триангуляции, поэтому разные триангуляции позволяют получить разные бордюры. Для завершения доказательства взаимно однозначного

соответствия нужно показать, что каждый $(m-1)$*-строчный узор бордюра, сложенный из* положительных целых чисел, можно получить на основе некоторой триангуляции.

Расширьте заданный произвольный узор из $m-1$ строк за счет вставки новой 0-й строки сверху и новой m-й строки снизу, которые состоят только из нулей. Обозначим теперь все элементы 0-й строки символами $(0,0)$, $(1,1)$, $(2,2)$ и т. д., а для всех неотрицательных целых чисел $u < v \le u+m$ предположим, что (u,v) — это элемент, направленный на юго-восток по диагонали от (u,u) и на юго-запад по диагонали от (v,v). Предполагается, что условие (**) выполняется для всех $u < v < u+m$. Действительно, его можно расширить до значительно более общего соотношения

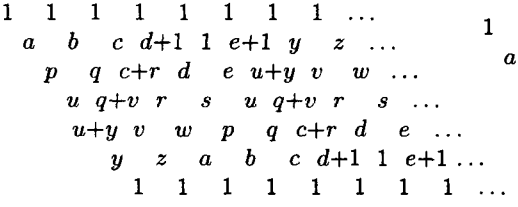
$$(t,u)(v,w) + (t,w)(u,v) = (t,v)(u,w) \qquad \text{для} \quad t \le u \le v \le w \le t+m. \qquad (***)$$

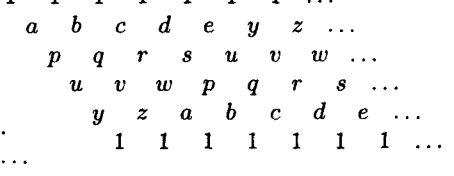
Если утверждение (***) ложно, пусть (t,u,v,w) — контрпример с наименьшим значением $(w-t)m + u - t + w - v$. *Случай 1.* $t+1 < u$. Тогда (***) выполняется для $(t,t+1,v,w)$, $(t,t+1,u,v)$ и $(t+1,u,v,w)$, поэтому получим $((t,u)(v,w) + (t,w)(v,u))(t+1,v) = (t,v)(u,w)(t+1,v)$; из этого следует, что $(t+1,v) = 0$, т. е. получили противоречие. *Случай 2.* $v+1 < w$. Тогда (***) выполняется для $(t,u,w-1,w)$, $(u,v,w-1,w)$ и $(t,u,v,w-1)$; снова получили аналогичное противоречие $(u,w-1) = 0$. *Случай 3.* $u = t+1$ и $w = v+1$. Теперь условие (***) сводится к условию (**).

Подставив $u = t+1$ и $w = t+m$ в (***), получим $(t,v) = (v,t+m)$ для $t \le v \le t+m$, так как $(t+1,t+m) = 1$ и $(t,t+m) = 0$. Итак, элементы любого $(m-1)$-строчного узора являются периодичными: $(u,v) = (v,u+m) = (u+m,v+m) = (v+m,u+2m) = \cdots$.

Каждый узор бордюра на основе положительных целых чисел содержит единицу во 2-й строке. Действительно, если положить $t = 0$, $v = u+1$ и $w = u+2$ в (***), то в результате получим $(0,u+1)(u,u+2) = (0,u)+(0,u+2)$; следовательно $(0,u+2)-(0,u+1) \ge (0,u+1)-(0,u)$ тогда и только тогда, когда $(u,u+2) \ge 2$. Это свойство выполняется не для всех u в диапазоне $0 \le u \le m-2$, поскольку $(0,1)-(0,0) = 1$ и $(0,m)-(0,m-1) = -1$.

Наконец, если $m > 3$, то во 2-й строке нельзя последовательно расположить две единицы, потому что из $(u,u+2) = (u+1,u+3) = 1$ следует, что $(u,u+3) = 0$. Значит, *бордюр с* m *строками можно свести к другому бордюру, в котором содержится на одну* строку меньше. Ниже показан пример приведения бордюра с семью строками к бордюру на основе шести строк.





Приведенный бордюр соответствует некой триангуляции, что доказывается по индукции, а неприведенный бордюр соответствует присоединению к ней еще одного треугольника. [*Math. Gazette* **57** (1974), 87–94, 175–183; Conway and Guy , *The Book of Numbers* (New York: Copernicus, 1996), 74–76, 96–97, 101–102.]

Замечания. Это доказательство демонстрирует, что функция (u,v), определенная на некоторой триангуляции с помощью матриц размера 2×2, удовлетворяет условию (***) всякий раз, когда (t,u,v,w) являются сторонами многоугольника в порядке обхода по часовой стрелке. Каждую функцию (u,v) можно представить в виде полинома относительно чисел $a_j = (j-1,j+1)$. Эти полиномы идентичны континуантам из раздела 4.5.3, за исключением знаков отдельных членов. Действительно, $(j,k) = i^{1-k+j}K_{k-j-1}(ia_{j+1}, ia_{j+2}, \ldots, ia_{k-1})$. Таким образом, (***) эквивалентно тождеству Эйлера для континуантов в ответе к

упр. 4.5.3–32. Матрицы L и R обладают интересным свойством: любая матрица неотрицательных целых чисел размера 2×2 с детерминантом, равным 1, может быть представлена единственным образом в виде произведения L и R.

Существует также несколько других интересных соотношений, например числа в строке 2 целочисленного бордюра обозначают количество треугольников, которые касаются каждой вершины соответствующего триангулированного многоугольника. Общее количество случаев, когда $(u.v) = 1$ в основной области $0 \le u < v - 1 < m - 1$ и $(u, v) \ne (0, m - 1)$, равно количеству диагоналей (хорд) триангуляции, а именно — $m - 3 = n - 1$. Общее количество двоек также равно $n - 1$, поскольку $(u, v) = 2$ тогда и только тогда, когда u_- и v_- являются противоположными вершинами двух треугольников, смежных с хордой.

Еще одну интерпретацию функции (u, v) предложили Д. М. Бролин (D. M. Broline), Д. У. Кроу (D. W. Crowe) и И. М. Айзекс (I. M. Isaacs) [*Geometriæ Dedicata* **3** (1974), 171–176]. Значение этой функции равно числу способов, с помощью которых можно установить соответствие для $v - u - 1$ вершин между ребрами u и $v - 1$ с различными треугольниками, смежными с этими вершинами.

РАЗДЕЛ 2.3.5

1. Структура Списка представляет собой ориентированный граф, в котором выходящие из вершин дуги упорядочены и некоторые вершины с нулевой степенью выхода обозначены как атомы. Более того, есть такая вершина S, что существует ориентированный путь от S к V для всех вершин $V \ne S$. (Если обратить направления дуг, то S станет корнем.)

2. Не совсем так, поскольку связи-нити в обычном представлении ведут к узлу-родителю `PARENT`, который не является единственным для подСписков. Возможно, для этого можно использовать предложенную в упр. 2.3.4.2–25 идею или аналогичный ей метод (но эта идея еще не применялась во время написания настоящей книги).

3. Как уже упоминалось в этом разделе, докажем, что `P` = `P0` по окончании выполнения алгоритма. Если нужно маркировать только узел `P0`, алгоритм, определенно, работает корректно. Если нужно маркировать $n > 1$ узлов, имеем `ATOM(P0)` = 0. Тогда на шаге E4 `ALINK(P0)` $\leftarrow \Lambda$ и алгоритм выполняется с `P0`, который заменяется на `ALINK(P0)`, и с `T`, который заменяется на `P0`. Согласно методу индукции (обратите внимание, что, так как `MARK(P0)` теперь равен 1, все связи с `P0` эквивалентны Λ во время выполнения шагов E4 и E5) приходим к выводу, что в конце концов будут маркированы все узлы на путях, которые начинаются с `ALINK(P0)` и не проходят через `P0`. В таком случае при переходе к шагу E6 получим `T` = `P0` и `P` = `ALINK(P0)`. Теперь, поскольку `ATOM(T)` = 1, на шаге E6 восстанавливаются значения `ALINK(P0)` и `ATOM(P0)` и совершается переход к шагу E5. На шаге E5 `BLINK(P0)` $\leftarrow \Lambda$ и т. д , причем согласно аналогичным доводам получим, что в конце концов будут маркированы все узлы на путях, которые начинаются с `BLINK(P0)` и не проходят через `P0`, или узлы, до которых можно добраться, начиная с `ALINK(P0)`. Затем совершается переход к шагу E6 со значениями `T` = `P0`, `P` = `BLINK(P0)`. В конечном счете при переходе к шагу E6 получается `T` = Λ, `P` = `P0`.

4. В приведенной ниже программе используются усовершенствованные приемы ускоренной обработки атомов, которые упомянуты в этом разделе сразу после описания алгоритма E.

На шагах E4 и E5 данного алгоритма необходимо проверить условие `MARK(Q)` = 0 Если `NODE(Q)` = +0, то этот особый случай можно соответствующим образом обработать, используя вместо него значение −0 и рассматривая его так, как если бы в самом начале оно было равно −0, поскольку обе связи (`ALINK` и `BLINK`) равны Λ в этом узле. Такое упрощение никак не повлияет на приведенную ниже оценку времени выполнения данной программы.

rI1 ≡ P, rI2 ≡ T, rI3 ≡ Q и rX ≡ −1 (для установки маркировочных битов MARK).

```
01  MARK  EQU  0:0
02  ATOM  EQU  1:1
03  ALINK EQU  2:3
04  BLINK EQU  4:5
05  E1    LD1  P0              1      E1. Инициализация. P ← P0.
06        ENT2 0               1      T ← Λ.
07        ENTX -1              1      rX ← −1.
08  E2    STX  0,1(MARK)       1      E2. Маркировка. MARK(P) ← 1.
09  E3    LDA  0,1(ATOM)       1      E3. Атом?
10        JAZ  E4              1      Выполнить переход, если ATOM(P) = 0.
11  E6    J2Z  DONE            n      E6. Вверх.
12        ENT3 0,2           n − 1    Q ← T.
13        LDA  0,3(ATOM)     n − 1
14        JANZ 1F            n − 1    Выполнить переход, если ATOM(T) = 1.
15        LD2  0,3(BLINK)      t2     T ← BLINK(Q).
16        ST1  0,3(BLINK)      t2     BLINK(Q) ← P.
17        ENT1 0,3             t2     P ← Q.
18        JMP  E6              t2
19  1H    STZ  0,2(ATOM)       t1     ATOM(T) ← 0.
20        LD2  0,3(ALINK)      t1     T ← ALINK(Q).
21        ST1  0,3(ALINK)      t1     ALINK(Q) ← P.
22        ENT1 0,3             t1     P ← Q.
23  E5    LD3  0,1(BLINK)      n      E5. Вниз по связям BLINK. Q ← BLINK(P).
24        J3Z  E6              n      Выполнить переход, если Q = Λ.
25        LDA  0,3           n − b2
26        STX  0,3(MARK)     n − b2   MARK(Q) ← 1.
27        JANP E6            n − b2   Выполнить переход, если узел
                                          NODE(Q) уже маркирован.
28        LDA  0,3(ATOM)     t2 + a2
29        JANZ E6            t2 + a2  Выполнить переход, если ATOM(Q) = 1.
30        ST2  0,1(BLINK)      t2     BLINK(P) ← T.
31  E4A   ENT2 0,1           n − 1    T ← P.
32        ENT1 0,3           n − 1    P ← Q.
33  E4    LD3  0,1(ALINK)      n      E4. Вниз по связям ALINK. Q ← ALINK(P).
34        J3Z  E5              n      Выполнить переход, если Q = Λ.
35        LDA  0,3           n − b1
36        STX  0,3(MARK)     n − b1   MARK(Q) ← 1.
37        JANP E5            n − b1   Выполнить переход, если узел
                                          NODE(Q) уже маркирован.
38        LDA  0,3(ATOM)     t1 + a1
39        JANZ E5            t1 + a1  Выполнить переход, если ATOM(Q) = 1.
40        STX  0,1(ATOM)       t1     ATOM(P) ← 1.
41        ST2  0,1(ALINK)      t1     ALINK(P) ← T.
42        JMP  E4A             t1     T ← P, P ← Q, переход к шагу E4. ▮
```

По закону Кирхгофа $t_1+t_2+1 = n$. Общее время выполнения равно $(34n+4t_1+3a-5b-8)u$, где n — количество маркированных узлов, не являющихся атомами, a — количество маркированных атомов, b — количество связей Λ, которые встречаются в маркированных узлах, не являющихся атомами, t_1 — число переходов вниз по связям ALINK ($0 \le t_1 < n$).

5. (Приведенный ниже алгоритм является самым быстрым из известных до сих пор алгоритмов маркировки для одноуровневой памяти.)

S1. Установить MARK(P0) $\leftarrow$ 1. Если ATOM(P0) = 1, то выполнение алгоритма прекращается; в противном случае установить S $\leftarrow$ 0, R $\leftarrow$ P0, T $\leftarrow \Lambda$.

S2. Установить P $\leftarrow$ BLINK(R). Если P = Λ или MARK(P) = 1, перейти к шагу S3. В противном случае установить MARK(P) $\leftarrow$ 1. Теперь, если ATOM(P) = 1, перейти к шагу S3; в противном случае, если S < N, установить S $\leftarrow$ S + 1, STACK[S] $\leftarrow$ P и перейти к шагу S3; в противном случае перейти к шагу S5.

S3. Установить P $\leftarrow$ ALINK(R). Если P = Λ или MARK(P) = 1, то перейти к шагу S4. В противном случае установить MARK(P) $\leftarrow$ 1. Теперь, если ATOM(P) = 1, перейти к шагу S4; в противном случае установить R $\leftarrow$ P и вернуться к шагу S2.

S4. Если S = 0, то прекратить выполнение алгоритма; в противном случае установить R $\leftarrow$ STACK[S], S $\leftarrow$ S $-$ 1 и перейти к шагу S2.

S5. Установить Q $\leftarrow$ ALINK(P). Если Q = Λ или MARK(Q) = 1, то перейти к шагу S6. В противном случае установить MARK(Q) $\leftarrow$ 1. Теперь, если ATOM(Q) = 1, перейти к шагу S6; в противном случае установить ATOM(P) $\leftarrow$ 1, ALINK(P) $\leftarrow$ T, T $\leftarrow$ P, P $\leftarrow$ Q и перейти к шагу S5.

S6. Установить Q $\leftarrow$ BLINK(P). Если Q = Λ или MARK(Q) = 1, то перейти к шагу S7; в противном случае установить MARK(Q) $\leftarrow$ 1. Теперь, если ATOM(Q) = 1, перейти к шагу S7; в противном случае установить BLINK(P) $\leftarrow$ T, T $\leftarrow$ P, P $\leftarrow$ Q и перейти к шагу S5.

S7. Если T = Λ, то перейти к шагу S3. В противном случае установить Q $\leftarrow$ T. Если ATOM(Q) = 1, то установить ATOM(Q) $\leftarrow$ 0, T $\leftarrow$ ALINK(Q), ALINK(Q) $\leftarrow$ P, P $\leftarrow$ Q и вернуться к шагу S6. Если ATOM(Q) = 0, то установить T $\leftarrow$ BLINK(Q), BLINK(Q) $\leftarrow$ P, P $\leftarrow$ Q и вернуться к шагу S7. ▮

[*См. CACM* **10** (1967), 501–506.]

6. Он включен в связи с выполнением второй фазы сборки мусора (или начальной фазы, если все маркировочные биты инициализируются в это время нулями).

7. Следует удалить шаги E2 и E3, а также команду "ATOM(P) $\leftarrow$ 1" на шаге E4. Установить MARK(P) $\leftarrow$ 1 на шаге E5 и использовать команду "MARK(Q) = 0", "MARK(Q) = 1" на шаге E6 вместо команд "ATOM(Q) = 1", "ATOM(Q) = 0" соответственно. Основная идея здесь заключается в установке маркировочного бита MARK только после маркирования левого поддерева. Этот алгоритм вполне пригоден, даже если дерево имеет перекрывающиеся (совместно используемые) поддеревья, но его нельзя применить для всех рекурсивных структур Списка, в которых узел NODE(ALINK(Q)) предшествует узлу NODE(Q). (Обратите внимание, что связь ALINK маркированного узла никогда не меняется.)

8. *Решение 1.* Оно аналогично алгоритму E, но проще.

F1. Установить T $\leftarrow \Lambda$, P $\leftarrow$ P0.

F2. Установить MARK(P) $\leftarrow$ 1 и P $\leftarrow$ P + SIZE(P).

F3. Если MARK(P) = 1, перейти к шагу F5.

F4. Установить Q $\leftarrow$ LINK(P). Если Q $\neq \Lambda$ и MARK(Q) = 0, то установить LINK(P) $\leftarrow$ T, T $\leftarrow$ P, P $\leftarrow$ Q и перейти к шагу F2. В противном случае установить P $\leftarrow$ P $-$ 1 и вернуться к шагу F3.

F5. Если T = Λ, прекратить выполнение алгоритма. В противном случае установить Q $\leftarrow$ T, T $\leftarrow$ LINK(Q), LINK(Q) $\leftarrow$ P, P $\leftarrow$ Q $-$ 1 и вернуться к шагу F3. ▮

Аналогичный алгоритм, который иногда позволяет сократить связанные с выделением памяти накладные расходы и избежать размещения всех указателей внутри узлов, предложил Ларс-Эрик Торелли (Lars-Erik Thorelli); см. *BIT* **12** (1972), 555–568.

Решение 2. Оно аналогично алгоритму D. В этом решении предполагается, что поле SIZE достаточно велико и в нем можно разместить адрес связи. Такое допущение, вероятно, не оправдано в данной постановке задачи, но оно позволяет получить несколько более быстрый алгоритм, чем алгоритм из первого решения.

G1. Установить $T \leftarrow \Lambda$, $\texttt{MARK(P0)} \leftarrow 1$. $P \leftarrow \texttt{P0} + \texttt{SIZE(P0)}$.

G2. Если $\texttt{MARK(P)} = 1$, перейти к шагу G5.

G3. Установить $Q \leftarrow \texttt{LINK(P)}$, $P \leftarrow P - 1$.

G4. Если $Q \neq \Lambda$ и $\texttt{MARK(Q)} = 0$, то установить $\texttt{MARK(Q)} \leftarrow 1$, $S \leftarrow \texttt{SIZE(Q)}$, $\texttt{SIZE(Q)} \leftarrow T$, $T \leftarrow Q + S$. Вернуться к шагу G2.

G5. Если $T = \Lambda$, прекратить выполнение алгоритма. В противном случае установить $P \leftarrow T$ и найти первое значение среди $Q = P, P-1, P-2, \ldots$, для которого $\texttt{MARK(Q)} = 1$; установить $T \leftarrow \texttt{SIZE(Q)}$ и $\texttt{SIZE(Q)} \leftarrow P - Q$. Вернуться к шагу G2. ▮

9. H1. Установить $L \leftarrow 0$, $K \leftarrow M + 1$, $\texttt{MARK(0)} \leftarrow 1$, $\texttt{MARK(M + 1)} \leftarrow 0$.

H2. Увеличить L на единицу и, если $\texttt{MARK(L)} = 1$, повторить этот шаг.

H3. Уменьшить K на единицу и, если $\texttt{MARK(K)} = 0$, повторить этот шаг.

H4. Если $L > K$, то перейти к шагу H5; в противном случае установить $\texttt{NODE(L)} \leftarrow \texttt{NODE(K)}$, $\texttt{ALINK(K)} \leftarrow L$, $\texttt{MARK(K)} \leftarrow 0$ и вернуться к шагу H2.

H5. Для $L = 1, 2, \ldots, K$ выполнить такие действия. Установить $\texttt{MARK(L)} \leftarrow 0$. Если $\texttt{ATOM(L)} = 0$ и $\texttt{ALINK(L)} > K$, то установить $\texttt{ALINK(L)} \leftarrow \texttt{ALINK(ALINK(L))}$. Если $\texttt{ATOM(L)} = 0$ и $\texttt{BLINK(L)} > K$, то установить $\texttt{BLINK(L)} \leftarrow \texttt{ALINK(BLINK(L))}$. ▮

[См. также упр. 2.5–33.]

10. Z1. [Инициализация.] Установить $F \leftarrow \texttt{P0}$, $R \Leftarrow \texttt{AVAIL}$, $\texttt{NODE(R)} \leftarrow \texttt{NODE(F)}$, $\texttt{REF(F)} \leftarrow R$. (Здесь F и R — указатели очереди в полях REF всех встретившихся узлов-заголовков.)

Z2. [Начало нового Списка.] Установить $P \leftarrow F$, $Q \leftarrow \texttt{REF(P)}$.

Z3. [Продвижение вправо.] Установить $P \leftarrow \texttt{RLINK(P)}$. Если $P = \Lambda$, то перейти к шагу Z6.

Z4. [Копирование одного узла.] Установить $\texttt{Q1} \Leftarrow \texttt{AVAIL}$, $\texttt{RLINK(Q)} \leftarrow \texttt{Q1}$, $Q \leftarrow \texttt{Q1}$, $\texttt{NODE(Q)} \leftarrow \texttt{NODE(P)}$.

Z5. [Преобразование связи подСписка.] Если $\texttt{T(P)} = 1$, то установить $\texttt{P1} \leftarrow \texttt{REF(P)}$ и, если $\texttt{REF(P1)} = \Lambda$, установить $\texttt{REF(R)} \leftarrow \texttt{P1}$, $R \Leftarrow \texttt{AVAIL}$, $\texttt{REF(P1)} \leftarrow R$, $\texttt{NODE(R)} \leftarrow \texttt{NODE(P1)}$, $\texttt{REF(Q)} \leftarrow R$. Если $\texttt{T(P)} = 1$ и $\texttt{REF(P1)} \neq \Lambda$, то установить $\texttt{REF(Q)} \leftarrow \texttt{REF(P1)}$. Перейти к шагу Z3.

Z6. [Переход к следующему списку.] Установить $\texttt{RLINK(Q)} \leftarrow \Lambda$. Если $\texttt{REF(F)} \neq R$, то установить $F \leftarrow \texttt{REF(REF(F))}$ и вернуться к шагу Z2. В противном случае установить $\texttt{REF(R)} \leftarrow \Lambda$, $P \leftarrow \texttt{P0}$.

Z7. [Окончательная зачистка.] Установить $Q \leftarrow \texttt{REF(P)}$. Если $Q \neq \Lambda$, то установить $\texttt{REF(P)} \leftarrow \Lambda$ и $P \leftarrow Q$ и повторить шаг Z7. ▮

Конечно, такое применение полей REF не позволяет использовать метод сборки мусора с помощью алгоритма D; более того, алгоритм D непригоден здесь еще и потому, что во время копирования не сохраняется структура Списков.

Несколько элегантных алгоритмов копирования и перемещения Списков основаны на гораздо более слабых предположениях о представлении Списка; см. D. W. Clark, *CACM* **19** (1976), 352–354; J. M. Robson, *CACM* **20** (1977), 431–433.

11. Выполним это упражнение вручную с помощью карандаша и бумаги, хотя можно было бы привести и более формальное решение. Сначала зададим уникальное имя (например, с помощью заглавных букв) для каждого Списка рассматриваемого множества. В данном примере получим $A = (a: C, b, a: F)$, $F = (b: D)$, $B = (a: F, b, a: E)$, $C = (b: G)$, $G = (a: C)$, $D = (a: F)$, $E = (b: G)$. Теперь построим список из пар имен Списков, эквивалентность которых необходимо доказать. Последовательно будем добавлять пары в этот список до тех пор, пока не получим противоречие из-за наличия противоречивых пар на первом уровне (тогда исходные Списки не эквивалентны), или до тех пор, пока из списка пар не будут следовать никакие другие пары (тогда исходные Списки не эквивалентны). В данном примере список пар в исходном состоянии содержит только заданную пару, AB; затем в него будут добавлены пары CF, EF (при сравнении A и B), DG (следует из CF), после чего получится непротиворечивое множество.

Для доказательства корректности этого метода обратите внимание на то, что (i) при получении ответа "не эквивалентны" заданные Списки не эквивалентны; (ii) если заданные Списки не эквивалентны, то будет получен ответ "не эквивалентны"; (iii) работа алгоритма всегда завершается.

12. Если список `AVAIL` содержит N узлов, где N — константа, выбор которой обсуждается ниже, то следует вызвать другую сопрограмму, которая использует вычислительные ресурсы вместе с основной программой и выполняет следующие действия: (a) маркирует все N узлов списка `AVAIL`; (b) маркирует другие узлы, которые свободны для этой программы; (c) связывает все немаркированные узлы вместе для подготовки нового списка `AVAIL`, который будет использоваться при опустошении текущего списка `AVAIL`; (d) сбрасывает маркировочные биты во всех узлах. Нужно выбрать такое N и такое отношение разделения времени, чтобы операции (a)–(d) гарантированно завершались до того, как N узлов будут извлечены из списка `AVAIL`, даже если основная программа выполняется достаточно быстро. При этом на этапе (b) необходимо убедиться в том, что маркированы все узлы, "доступные для данной программы", по мере ее выполнения; остальные подробности здесь опускаются. Если в созданном на этапе (c) списке содержится менее N узлов, то в конце концов может потребоваться прекратить работу приложения из-за того, что в памяти будет исчерпано все свободное пространство. Более подробные сведения по этой теме приводятся в работах Guy L. Steele Jr., *CACM* **18** (1975), 495–508; P. Wadler, *CACM* **19** (1976), 491–500; E. W. Dijkstra, L. Lamport, A. J. Martin, C. S. Scholten, and E. F. M. Steffens, *CACM* **21** (1978), 966–975; H. G. Baker, Jr., *CACM* **21** (1978), 280–294.

РАЗДЕЛ 2.4

1. В прямом порядке.

2. Оно прямо пропорционально количеству создаваемых элементов таблицы данных.

3. Заменить операции шага A5 такими действиями.

A5′. [Удаление верхнего уровня.] Удалить верхний элемент стека; и если номер нового уровня на вершине стека равен $\geq$ `L`, то пусть (`L1`, `P1`) — новый элемент на вершине стека. Повторить этот шаг. В противном случае установить `SIB(P1)` $\leftarrow$ `Q`. Тогда пусть (`L1`, `P1`) является новым элементом на вершине стека.

4. (Решение Дэвида С. Вайса (David S. Wise).) Правило (c) нарушается тогда и только тогда, когда существует элемент данных, *полная квалификация* A_0 `OF` ... `OF` A_n которого в языке COBOL также является ссылкой на некоторый элемент данных. Так как и родитель

A_1 OF ... OF A_n должен удовлетворять правилу (c), можно предположить, что этот другой элемент данных является наследником некоторого родителя. Значит, алгоритм A следует расширить таким образом, чтобы при добавлении в таблицу данных каждого нового элемента данных выполнялась проверка, является ли его родитель предком любого другого элемента данных с таким же именем или есть ли в стеке родитель любого другого элемента с таким же именем. (Когда родитель равен Λ, он является предком всех других элементов и всегда располагается в стеке.)

С другой стороны, если оставить алгоритм A в прежнем состоянии, программист при выполнении COBOL-программы получит сообщение об ошибке со стороны алгоритма B при попытке использования недопустимого элемента. И только при выполнении команды MOVE CORRESPONDING элементы данных могут применяться без вывода сообщения об ошибке.

5. Следует выполнить такие изменения.

На шаге	*команду*	*заменить командой*
B1	P $\leftarrow$ LINK(P_0)	P $\leftarrow$ LINK(INFO(T))
B2	$k \leftarrow 0$	K $\leftarrow$ T
B3	$k < n$	RLINK(K) $\neq \Lambda$
B4	$k \leftarrow k + 1$	K $\leftarrow$ RLINK(K)
B6	NAME(S) $= P_k$	NAME(S) = INFO(K)

6. Простое изменение алгоритма B позволяет выполнять поиск только полных ссылок (если $k = n$ и PARENT(S) $\neq \Lambda$ на шаге B3 или если NAME(S) $\neq P_k$ на шаге B6, то в таком случае следует установить P $\leftarrow$ PREV(P) и перейти к шагу B2). Основная идея здесь заключается в том, чтобы сначала выполнить модифицированную версию алгоритма B, а затем, если ссылка Q *все еще* равна Λ, — его немодифицированную версию.

7. MOVE MONTH OF DATE OF SALES TO MONTH OF DATE OF PURCHASES. MOVE DAY OF DATE OF SALES TO DAY OF DATE OF PURCHASES. MOVE YEAR OF DATE OF SALES TO YEAR OF DATE OF PURCHASES. MOVE ITEM OF TRANSACTION OF SALES TO ITEM OF TRANSACTION OF PURCHASES. MOVE QUANTITY OF TRANSACTION OF SALES TO QUANTITY OF TRANSACTION OF PURCHASES. MOVE PRICE OF TRANSACTION OF SALES TO PRICE OF TRANSACTION OF PURCHASES. MOVE TAX OF TRANSACTION OF SALES TO TAX OF TRANSACTION OF PURCHASES.

8. Тогда и только тогда, когда α или β является простейшим элементом. (Может быть, следует отметить, что автору не удалось должным образом обработать этот случай в первом черновом варианте алгоритма C, что существенно усложнило этот алгоритм.)

9. Если ни α, ни β не является простейшим элементом, то команда MOVE CORRESPONDING α TO β эквивалентна набору команд MOVE CORRESPONDING A OF α TO A OF β, выполненному над всеми именами A, общими для групп α и β. (Этот способ определения элегантнее, чем более традиционное и тяжеловесное определение команды MOVE CORRESPONDING, которое предлагается в тексте.) Можно убедиться, что алгоритм C удовлетворяет такому определению, доказав по индукции, что после выполнения шагов C2–C5 в конце концов получится P = P0 и Q = Q0. Далее доказательство проводится точно так, как оно выполнялось много раз прежде, в "индукции по дереву" (см. например, доказательство алгоритма 2.3.1T).

10. (a) Установить S1 $\leftarrow$ LINK(P_k). Затем повторно ни разу не устанавливать или устанавливать S1 $\leftarrow$ PREV(S1) до тех пор, пока либо S1 $= \Lambda$ (NAME(S) $\neq P_k$), либо S1 = S (NAME(S) $= P_k$). (b) Установить P1 $\leftarrow$ P, а затем ни разу не устанавливать или устанавливать P1 $\leftarrow$ PREV(P1) до тех пор, пока не выполнится условие PREV(P1) $= \Lambda$. Выполнить аналогичную операцию для переменных Q1 и Q; затем проверить P1 = Q1. Иначе, если позиции таблицы данных упорядочены так, что PREV(P) $<$ P для всех P, более быструю проверку, очевидно, можно выполнить следующим образом: выяснить, что P $>$ Q, или,

наоборот, пройти по связям PREV большего из них и обнаружить, встречается ли меньший из них.

11. Незначительного повышения скорости выполнения шага C4 можно достичь за счет добавления нового поля связи SIB1(P) ≡ CHILD(PARENT(P)). Увеличение скорости будет более существенным, если модифицировать связи CHILD и SIB таким образом, чтобы NAME(SIB(P)) > NAME(P). Это позволит значительно ускорить поиск на шаге C3, поскольку для этого потребуется выполнить только один проход по каждому семейству для поиска подобных членов Следовательно, такое изменение позволяет исключить единственную операцию поиска в алгоритме C. Алгоритмы A и C можно легко модифицировать с предложенной интерпретацией, а читателю будет полезно выполнить ее в качестве интересного упражнения. (Однако, если рассмотреть относительную частоту команд MOVE CORRESPONDING и обычный размер семейных групп, полученное в результате повышение скорости будет не таким уж значительным после трансляции реальных COBOL-программ.)

12. Шаги B1–B3 остаются прежними, а другие шаги будут выглядеть так

B4. Установить $k \leftarrow k + 1$, R $\leftarrow$ LINK(P_k).

B5. Если R = Λ, то соответствия нет; установить P ← PREV(P) и перейти к шагу B2. Если R < S ≤ SCOPE(R), установить S ← R и перейти к шагу B3. В противном случае установить R ← PREV(R) и повторить шаг B5. ▮

Этот алгоритм *не* адаптирован для соглашений, принятых для языка PL/I в упр. 6.

13. Используйте тот же алгоритм, но без операций установки связей NAME, PARENT, CHILD и SIB. При каждом удалении верхнего элемента стека на шаге A5 следует устанавливать SCOPE(P1) ← Q − 1. После исчерпания входного потока на шаге A2 следует просто установить L ← 0 и продолжить выполнение алгоритма. Затем необходимо прекратить выполнение алгоритма, если L = 0 на шаге A7.

14. Шаги приведенного ниже алгоритма со вспомогательным стеком пронумерованы так же, как шаги алгоритма C, приведенного в данном разделе, для упрощения их сравнительного анализа.

C1. Установить P ← P0, Q ← Q0 и опустошить стек.

C2. Если SCOPE(P) = P или SCOPE(Q) = Q, вывести (P, Q) как одну из искомых пар и перейти к шагу C5. В противном случае поместить (P, Q) в стек и установить P ← P + 1, Q ← Q + 1.

C3. Определить, указывают ли P и Q на элементы с тем же именем (см. упр 10, (b)). Если указывают, то перейти к шагу C2. Если не указывают, то поместить (P1, Q1) в верхнюю часть стека; если SCOPE(Q) < SCOPE(Q1), то установить Q ← SCOPE(Q) + 1 и повторить шаг C3

C4. Поместить (P1, Q1) в верхнюю часть стека. Если SCOPE(P) < SCOPE(P1), то установить P ← SCOPE(P)+1, Q ← Q1+1 и перейти к шагу C3. Если SCOPE(P) = SCOPE(P1), то установить P ← P1, Q ← Q1 и удалить верхний элемент стека.

C5. Если стек пуст, прекратить выполнение алгоритма. В противном случае перейти к шагу C4. ▮

РАЗДЕЛ 2.5

1. В столь благоприятных ситуациях можно воспользоваться стековыми операциями следующим образом. Пусть область пула памяти имеет адреса от 0 до M − 1 и пусть AVAIL указывает на нижний свободный адрес. При выделения N слов, если AVAIL + N ≥ M, сообщается о невозможности выделения необходимого количества памяти, в противном случае

устанавливается AVAIL ← AVAIL+N. Для освобождения этих N слов просто устанавливается AVAIL ← AVAIL − N.

Аналогично для метода "первым выделен — первым освобожден" используются операции с циклической очередью.

2. Количество памяти, необходимой для хранения элемента длиной l, составляет $k\lceil l/(k-b)\rceil$, среднее значение которого равно $kL/(k-b)+(1-\alpha)k$, где α предполагается равным $1/2$, не зависящим от k. Это выражение минимально (для действительных значений k) при $k=b+\sqrt{2bL}$. Поэтому выбор k равным ближайшему (сверху или снизу) к этой величине целому значению дает наименьшее значение $kL/(k-b)+\frac{1}{2}k$. Например, при $b=1$ и $L=10$ следует принять $k\approx 1+\sqrt{20}=5$ или 6; оба значения одинаково хороши. Более детально эта задача рассматривается в *JACM* **12** (1965), 53–70.

4. rI1 ≡ Q, rI2 ≡ P.

```
A1   LDA  N          rA ← N.
     ENT2 AVAIL      P ← LOC(AVAIL).
A2A  ENT1 0,2        Q ← P.
A2   LD2  0,1(LINK)  P ← LINK(Q).
     J2N  OVERFLOW   Если P = Λ, нет памяти.
A3   CMPA 0,2(SIZE)
     JG   A2A        Переход при N > SIZE(P).
A4   SUB  0,2(SIZE)  rA ← N − SIZE(P) ≡ K.
     JANZ *+3        Переход при K ≠ 0.
     LDX  0,2(LINK)
     STX  0,1(LINK)  LINK(Q) ← LINK(P).
     STA  0,2(SIZE)  SIZE(P) ← K.
     LD1  0,2(SIZE)  Возможное окончание
     INC1 0,2          с установкой rI1 ← P + K.   ▮
```

5. Вероятно, не стоит. Недоступная область памяти непосредственно перед адресом P станет впоследствии свободной, и ее длина увеличится на величину K; увеличение на 99 не является незначительным.

6. Идея заключается в поиске подходящего блока всякий раз в различных частях списка AVAIL. Например, можно воспользоваться блуждающим указателем (roving pointer) с именем ROVER, который работает следующим образом. На шаге A1 установить Q ← ROVER. После шага A4 установить ROVER ← LINK(Q), если LINK(Q) ≠ Λ; в противном случае установить ROVER ← LOC(AVAIL). На шаге A2, когда впервые при некотором выполнении алгоритма A P = Λ, установить Q ← LOC(AVAIL) и повторить шаг A2. Когда P = Λ во *второй раз*, алгоритм завершается неудачно. Таким образом, ROVER будет стремиться указывать на случайные места в списке AVAIL и размеры окажутся более сбалансированными. В начале программы установите ROVER ← LOC(AVAIL). Необходимо *также* устанавливать ROVER равным LOC(AVAIL) каждый раз, когда блок, адрес которого равен текущему значению ROVER, удаляется из списка AVAIL. (Впрочем, иногда полезно иметь в начале памяти блоки малого размера, как в случае метода первого подходящего; например, в конце памяти может храниться последовательный стек. В таких ситуациях можно уменьшить время поиска с помощью деревьев, как предложено в упр. 6.2.3–30.)

7. 2000, 1000 с запросами на выделение блоков размером 800, 1300.

[Пример, когда работает метод *наихудшего подходящего*, но не работает метод наилучшего подходящего, был предложен Р. Д. Вейландом (R. J. Weiland).]

8. На шаге A1 установить M ← ∞, R ← Λ. На шаге A2, если P = Λ, перейти к шагу A6. На шаге A3 перейти к шагу A5 (а не к шагу A4). Добавить следующие шаги.

A5. [Наилучший подходящий?] Если M > SIZE(P), установить R ← Q и M ← SIZE(P). Затем установить Q ← P и вернуться к шагу A2.

A6. [Найден ли блок?] Если R = Λ, алгоритм завершается неудачно. В противном случае установить Q ← R, P ← LINK(Q) и перейти к шагу A4. ▮

9. Очевидно, что если нам везет и мы находим блок с SIZE(P) = N, то найден наилучший подходящий блок и дальнейший поиск на этом прекращается. (Если используются блоки лишь нескольких разных размеров, это происходит достаточно часто.) Если применить метод "помеченной границы" наподобие использованного в алгоритме C, можно содержать список AVAIL в рассортированном по размерам блоков виде. При этом длительность поиска в среднем может быть снижена до половины длины списка (или даже до меньшей величины). Однако наилучшим решением представляется размещение списка AVAIL в виде структуры сбалансированного дерева, описываемого в разделе 6.2.3, в случае, если список имеет достаточно большой размер.

10. Необходимо сделать следующие изменения.

Шаг B2: вместо "P > P0" вставить "P ≥ P0".

Шаг B3: вставить "Если P0 + N > P и P ≠ Λ, установить P ← LINK(P) и повторить шаг B3".

Шаг B4: вместо "Q + SIZE(Q) = P0" вставить "Q + SIZE(Q) ≥ P0", а вместо "SIZE(Q) ← SIZE(Q) + N" — "SIZE(Q) ← P0 + N − Q".

11. Если P0 > ROVER, на шаге B1 вместо Q ← LOC(AVAIL) можно установить Q ← ROVER. Если в списке AVAIL имеется n элементов, среднее количество итераций на шаге B2 составляет $(2n+3)(n+2)/6(n+1) = \frac{1}{3}n + \frac{5}{6} + O\left(\frac{1}{n}\right)$. Например, если $n = 2$, будет получено 9 равновероятных ситуаций, в которых P1 и P2 указывают на два существующих свободных блока.

	P0 < P1	P1 < P0 < P2	P2 < P0
ROVER = P1	1	1	2
ROVER = P2	1	2	1
ROVER = LOC(AVAIL)	1	2	3

На этой диаграмме показано число итераций, необходимых в каждом случае. Среднее значение составляет

$$\frac{1}{9}\left(\binom{2}{2} + \binom{3}{2} + \binom{4}{2} + \binom{3}{2} + \binom{2}{2}\right) = \frac{1}{9}\left(\binom{5}{3} + \binom{4}{3}\right) = \frac{14}{9}.$$

12. A1. Установить P ← ROVER, F ← 0.

A2. Если P = LOC(AVAIL) и F = 0, установить P ← AVAIL, F ← 1 и повторить шаг A2. Если P = LOC(AVAIL) и F ≠ 0, алгоритм завершается неудачно.

A3. Если SIZE(P) ≥ N, перейти к шагу A4; в противном случае установить P ← LINK(P) и вернуться к шагу A2.

A4. Установить ROVER ← LINK(P), K ← SIZE(P) − N. Если K < c (где c — константа ≥ 2), установить LINK(LINK(P+1)) ← ROVER, LINK(ROVER+1) ← LINK(P+1), L ← P; в противном случае установить L ← P+K, SIZE(P) ← SIZE(L−1) ← K, TAG(L − 1) ← "−", SIZE(L) ← N. И наконец, установить TAG(L) ← TAG(L + SIZE(L) − 1) ← "+". ▮

13. rI1 ≡ P, rX ≡ F, rI2 ≡ L.

```
LINK  EQU  4:5
SIZE  EQU  1:2
TSIZE EQU  0:2
TAG   EQU  0:0
```

```
A1    LDA  N            rA ← N.
      SLA  3            Сдвиг в поле SIZE
      ENTX 0            F ← 0.
      LD1  ROVER        P ← ROVER.
      JMP  A2
A3    CMPA 0,1(SIZE)
      JLE  A4           Переход, если N ≤ SIZE(P).
      LD1  0,1(LINK)    P ← LINK(P).
A2    ENT2 -AVAIL,1     rI2 ← P − LOC(AVAIL).
      J2NZ A3
      JXNZ OVERFLOW     F ≠ 0?
      ENTX 1            Установить F ← 1.
      LD1  AVAIL(LINK)  P ← AVAIL.
      JMP  A2
A4    LD2  0,1(LINK)
      ST2  ROVER        ROVER ← LINK(P).
      LDA  0,1(SIZE)    rA ≡ K ← SIZE(P) − N.
      SUB  N
      CMPA =c=
      JGE  1F           Переход, если K ≥ c.
      LD3  1,1(LINK)    rI3 ← LINK(P + 1).
      ST2  0,3(LINK)    LINK(rI3) ← ROVER.
      ST3  1,2(LINK)    LINK(ROVER + 1) ← rI3.
      ENT2 0,1          L ← P
      LD3  0,1(SIZE)    rI3 ← SIZE(P).
      JMP  2F
1H    STA  0,1(SIZE)    SIZE(P) ← K.
      LD2  0,1(SIZE)
      INC2 0,1          L ← P + K.
      LDAN 0,1(SIZE)    rA ← −K.
      STA  -1,2(TSIZE)  SIZE(L − 1) ← K, TAG(L − 1) ← "−".
      LD3  N            rI3 ← N.
2H    ST3  0,2(TSIZE)   TAG(L) ← "+", установить также SIZE(L) ← rI3.
      INC3 0,2
      STZ  -1,3(TAG)    TAG(L + SIZE(L) − 1) ← "+".  ▮
```

14. (a) Это поле необходимо для определения начала блока на шаге C2. Его можно заменить (возможно, с преимуществом перед исходным расположением) связью с первым словом блока (см. также упр. 19) (b) Это поле необходимо, так как иногда нужно выделить более N слов (например, если K = 1) и при освобождении надо знать количество выделявшейся памяти.

15, 16. rI1 ≡ P0, rI2 ≡ P1, rI3 ≡ F, rI4 ≡ B, rI6 ≡ −N.

```
D1 LD1  P0           D1.
   LD2  0,1(SIZE)
   ENN6 0,2          N ← SIZE(P0).
   INC2 0,1          P1 ← P0 + N.
   LD5  0,2(TSIZE)
   J5N  D4           Перейти к шагу D4, если TAG(P1) = "−".
D2 LD5  -1,1(TSIZE)  D2.
   J5N  D7           Перейти к шагу D7, если TAG(P0 − 1) = "−".
```

```
D3 LD3   AVAIL(LINK)   D3. Установить F ← AVAIL.
   ENT4  AVAIL         B ← LOC(AVAIL).
   JMP   D5            Перейти к шагу D5.
D4 INC6  0,5           D4. N ← N + SIZE(P1).
   LD3   0,2(LINK)     F ← LINK(P1).
   LD4   1,2(LINK)     B ← LINK(P1 + 1).
   CMP2  ROVER         (Новый код, связанный со свойством
   JNE   *+3             ROVER из упр. 12.
   ENTX  AVAIL           Если P1 = ROVER,
   STX   ROVER             установить ROVER ← LOC(AVAIL).)
   DEC2  0,5           P1 ← P1 + SIZE(P1).
   LD5   -1,1(TSIZE)
   J5N   D6            Перейти к шагу D6, если TAG(P0 − 1) = "−".
D5 ST3   0,1(LINK)     D5. LINK(P0) ← F.
   ST4   1,1(LINK)     LINK(P0 + 1) ← B.
   ST1   1,3(LINK)     LINK(F + 1) ← P0.
   ST1   0,4(LINK)     LINK(B) ← P0.
   JMP   D8            Перейти к шагу D8.
D6 ST3   0,4(LINK)     D6. LINK(B) ← F.
   ST4   1,3(LINK)     LINK(F + 1) ← B.
D7 INC6  0,5           D7. N ← N + SIZE(P0 − 1).
   INC1  0,5           P0 ← P0 − SIZE(P0 − 1).
D8 ST6   0,1(TSIZE)    D8. SIZE(P0) ← N, TAG(P0) ← "−".
   ST6   -1,2(TSIZE)   SIZE(P1 − 1) ← N, TAG(P1 − 1) ← "−". ▮
```

17. Оба поля LINK равны LOC(AVAIL).

18. Алгоритм A выделяет верхнюю часть большого блока. Когда вся память доступна, метод первого подходящего начинает работу с выделения памяти в старших адресах, однако повторно после освобождения эти блоки памяти не резервируются, поскольку обычно подходящие свободные блоки находятся в младших адресах. Таким образом, большой свободный блок, расположенный в начале памяти, при использовании метода первого подходящего быстро исчезает. Однако большой блок редко оказывается наиболее подходящим, а потому метод наиболее подходящего оставляет большой блок в начале памяти.

19. Используйте алгоритм из упр. 12 с исключением из шага A4 ссылок на SIZE(L − 1), TAG(L − 1) и TAG(L + SIZE(L) − 1); вставьте также следующие новые шаги между A2 и A3.

A2$\frac{1}{2}$. Установить P1 ← P + SIZE(P). Если TAG(P1) = "+", перейти к шагу A3. В противном случае установить P2 ← LINK(P1), LINK(P2 + 1) ← LINK(P1 + 1), LINK(LINK(P1 + 1)) ← P2, SIZE(P) ← SIZE(P) + SIZE(P1). Если ROVER = P1, установить ROVER ← P2. Повторить шаг A2$\frac{1}{2}$.

Ясно, что ситуации (2)–(4) здесь встретиться не могут; единственный реальный эффект от такого распределения памяти заключается в тенденции к удлинению поиска по сравнению с упр. 12, а иногда в том, что K будет меньше, чем c, хотя реально будет существовать доступный блок, предшествующий данному, о котором нам ничего не известно.

(Имеется альтернативный вариант, при котором слияние блоков выносится из внутреннего цикла A3 и выполняется только на шаге A4 перед окончательным выделением памяти или во внутреннем цикле, если иначе алгоритм завершится неудачно. Такой альтернативный вариант требует модельного изучения для того, чтобы установить, имеются ли у него какие-либо преимущества.)

[Этот метод с небольшими усовершенствованиями, как было доказано, вполне удовлетворителен для реализации TEX и METAFONT. См. *TEX: The Program* (Addison-Wesley. 1986), §125.]

20. Когда обнаруживается свободный двойник, при выполнении цикла слияния необходимо удалить блок из списка AVAIL[k], однако неизвестно, какие связи следует обновить. Для получения ответа на этот вопрос можно (i) выполнить длинный поиск или (ii) сделать список двусвязным.

21. Если $n = 2^k\alpha$, где $1 \le \alpha \le 2$, a_n равно $2^{2k+1}(\alpha - \frac{2}{3}) + \frac{1}{3}$, а b_n равно $2^{2k-1}\alpha^2 + 2^{k-1}\alpha$. Отношение a_n/b_n для больших n, по сути, равно величине $4(\alpha - \frac{2}{3})/\alpha^2$, которая принимает минимальное значение $\frac{4}{3}$ при $\alpha = 1$ и 2 и максимальное значение $\frac{3}{2}$ при $\alpha = 1\frac{1}{3}$. Так что a_n/b_n не имеет предела, колеблясь между этими двумя крайними значениями. Методы усреднения из раздела 4.2.4, однако, дают нам среднее отношение, равное $4(\ln 2)^{-1} \int_1^2 (\alpha - \frac{2}{3})\, d\alpha/\alpha^3 = (\ln 2)^{-1} \approx 1.44$.

22. Для этой идеи необходимо поле TAG в нескольких словах блока из 11 слов, а не только в первом слове. Она работоспособна, если можно выделить дополнительные биты TAG, и, скорее всего, подходит, в первую очередь, для аппаратной реализации.

23. 011011110100; 011011100000.

24. Это привнесет ошибку в программу; можно оказаться на шаге S1 при TAG(0) = 1, поскольку из шага S2 можно вернуться к шагу S1. Для работоспособности программы необходимо на шаге S2 добавить "TAG(L) $\leftarrow$ 0" после "L $\leftarrow$ P". (Впрочем, проще считать, что TAG(2^m) = 0.)

25. Идея абсолютно верна (критика не всегда должна быть только отрицательной). Заголовки списков AVAIL[k] могут быть обрезаны для $n < k \le m$; приведенные в тексте раздела алгоритмы могут использоваться с заменой m на n на шагах R1 и S1. Начальные условия (13) и (14) должны быть изменены таким образом, чтобы указывалось 2^{m-n} блоков размером 2^n, а не один блок размером 2^m.

26. Используя двоичное представление M, можно легко изменить начальные условия (13) и (14) так, чтобы вся память была разделена на блоки, размеры которых представляют собой степени двойки, с блоками в порядке убывания размеров. В алгоритме S TAG(P) должен рассматриваться как 0 всякий раз, когда P $\ge$ M $- 2^k$.

27. rI1 $\equiv k$, rI2 $\equiv j$, rI3 $\equiv j - k$, rI4 $\equiv$ L, LOC(AVAIL[j]) = AVAIL + j.

Предполагаем, что имеется вспомогательная таблица TWO[j] = 2^j, хранящаяся в ячейках TWO + j, для $0 \le j \le m$. Предположим далее, что "+" и "$-$" представляют дескрипторы 0 и 1 и что TAG(LOC(AVAIL[j])) = "$-$", но в качестве признака конца памяти TAG(LOC(AVAIL[m+1])) = "+".

```
00  KVAL   EQU  5:5
01  TAG    EQU  0:0
02  LINKF  EQU  1:2
03  LINKB  EQU  3:4
04  TLNKF  EQU  0:2
05  R1     LD1  K                  1     R1. Поиск блока.
06         ENT2 0,1                1     j ← k.
07         ENT3 0                  1
08         LD4  AVAIL,2(LINKF)     1
09  1H     ENT5 AVAIL,2          1+R
10         DEC5 0,4              1+R
11         J5NZ R2               1+R     Переход при AVAILF[j] ≠ LOC(AVAIL[j]).
12         INC2 1                  R     Увеличение j.
13         INC3 1                  R
14         LD4N AVAIL,2(TLNKF)     R
15         J4NN 1B                 R     j ≤ m?
16         JMP  OVERFLOW
```

```
17 R2    LD5  0,4(LINKF)        1    R2. Удаление из списка.
18       ST5  AVAIL,2(LINKF)    1    AVAILF[j] ← LINKF(L).
19       ENTA AVAIL,2           1
20       STA  0,5(LINKB)        1    LINKB(L) ← LOC(AVAIL[j]).
21       STZ  0,4(TAG)          1    TAG(L) ← 0.
22 R3    J3Z  DONE              1    R3. Требуется разделение?
23 R4    DEC3 1                 R    R4. Разделение.
24       DEC2 1                 R    Уменьшение j.
25       LD5  TWO,2             R    rI5 ≡ P.
26       INC5 0,4               R    P ← L + 2^j.
27       ENNA AVAIL,2           R
28       STA  0,5(TLNKF)        R    TAG(P) ← 1, LINKF(P) ← LOC(AVAIL[j]).
29       STA  0,5(LINKB)        R    LINKB(P) ← LOC(AVAIL[j]).
30       ST5  AVAIL,2(LINKF)    R    AVAILF[j] ← P.
31       ST5  AVAIL,2(LINKB)    R    AVAILB[j] ← P.
32       ST2  0,5(KVAL)         R    KVAL(P) ← j.
33       J3P  R4                R    Переход к шагу R3.
34 DONE  ...                           ∎
```

28. rI1 ≡ k, rI5 ≡ P, rI4 ≡ L; полагаем TAG(2^m) = "+".

```
01 S1 LD4  L                   1      S1. Свободен ли двойник?
02    LD1  K                   1
03 1H ENTA 0,4                 1+S
04    XOR  TWO,1               1+S    rA ← двойник_k(L).
05    STA  TEMP                1+S
06    LD5  TEMP                1+S    P ← rA.
07    LDA  0,5                 1+S
08    JANN S3                  1+S    Переход при TAG(P) = 0.
09    CMP1 0,5(KVAL)           B+S
10    JNE  S3                  B+S    Переход при KVAL(P) ≠ k.
11 S2 LD2  0,5(LINKF)          S      S2. Объединение с двойником.
12    LD3  0,5(LINKB)          S
13    ST3  0,2(LINKF)          S      LINKF(LINKB(P)) ← LINKF(P).
14    ST2  0,3(LINKB)          S      LINKB(LINKF(P)) ← LINKB(P).
15    INC1 1                   S      Увеличение k.
16    CMP4 TEMP                S
17    JL   1B                  S
18    ENT4 0,5                 A      Если L > P, установить L ← P.
19    JMP  1B                  A
20 S3 LD2  AVAIL,1(LINKF)      1      S3. Помещение в список.
21    ENNA AVAIL,1             1
22    STA  0,4(0:4)            1      TAG(L) ← 1, LINKB(L) ← LOC(AVAIL[k]).
23    ST2  0,4(LINKF)          1      LINKF(L) ← AVAILF[k].
24    ST1  0,4(KVAL)           1      KVAL(L) ← k.
25    ST4  0,2(LINKB)          1      LINKB(AVAILF[k]) ← L.
26    ST4  AVAIL,1(LINKF)      1      AVAIL[k] ← L.   ∎
```

29. Может, но только за счет некоторого поиска или (что предпочтительнее) дополнительной каким-либо образом упакованной таблицы битов TAG. (Заманчиво объединить двойников не в алгоритме S, а только в алгоритме R, если нет достаточно большого блока, чтобы отвечать запросу. Но, вероятно, это вызовет повышенную фрагментацию памяти.)

31. См. David L. Russell, *SICOMP* **6** (1977), 607–621.

33. **G1.** [Очистка полей связей.] Установить P ← 1 и повторять операции LINK(P) ← Λ и P ← P + SIZE(P) до тех пор, пока не будет выполнено условие P = AVAIL. (Таким образом поля LINK в первых словах каждого узла устанавливаются равными Λ. В большинстве случаев можно считать, что этот шаг не является необходимым, *поскольку поле* LINK(P) *устанавливается равным* Λ *на шаге* G9 *ниже*, а также может быть сброшено распределителем памяти.)

G2. [Инициализация фазы маркировки.] Установить TOP ← USE, LINK(TOP) ← AVAIL, LINK(AVAIL) ← Λ. (TOP указывает на вершину стека, как в алгоритме 2.3.5D.)

G3. [Поднятие стека.] Установить P ← TOP, TOP ← LINK(TOP). Если TOP = Λ, перейти к шагу G5.

G4. [Поместить новые связи в стек.] Для $1 \le k \le$ T(P) выполнить следующие операции: установить Q ← LINK(P + k); затем, если Q ≠ Λ и LINK(Q) = Λ, установить LINK(Q) ← TOP, TOP ← Q, после чего вернуться к шагу G3.

G5. [Инициализация следующей фазы.] (Теперь P = AVAIL и фаза маркировки завершена, так что первое слово каждого доступного узла имеет ненулевую связь LINK. Цель наших последующих действий — объединение смежных недоступных узлов для ускорения последующих шагов и назначение новых адресов доступным узлам.) Установить Q ← 1, LINK(AVAIL) ← Q, SIZE(AVAIL) ← 0, P ← 1. (Ячейка AVAIL используется в качестве ограничителя для указания конца цикла в последующих фазах.)

G6. [Присвоение новых адресов.] Если LINK(P) = Λ, перейти к шагу G7. Если условие не соблюдено, то при SIZE(P) = 0 перейти к шагу G8, иначе — установить LINK(P) ← Q, Q ← Q + SIZE(P), P ← P + SIZE(P) и повторить этот шаг.

G7. [Слияние свободных областей.] Если LINK(P + SIZE(P)) = Λ, увеличить SIZE(P) на SIZE(P + SIZE(P)) и повторить этот шаг. Иначе — установить P ← P + SIZE(P) и вернуться к шагу G6.

G8. [Преобразование всех связей.] (Теперь в поле LINK первого слова каждого доступного узла содержится адрес, куда будет перемещен узел.) Установить USE ← LINK(USE) и AVAIL ← Q. Затем установить P ← 1 и повторять следующие операции до тех пор, пока не будет выполнено условие SIZE(P) = 0: если LINK(P) ≠ Λ, установить LINK(Q) ← LINK(LINK(Q)) для всех Q, таких, что P < Q ≤ P + T(P) и LINK(Q) ≠ Λ; затем независимо от значения LINK(P) установить P ← P + SIZE(P).

G9. [Перемещение.] Установить P ← 1 и повторять следующие операции до тех пор, пока не будет выполнено условие SIZE(P) = 0: установить Q ← LINK(P) и, если Q ≠ Λ, установить LINK(P) ← Λ и NODE(Q) ← NODE(P); затем независимо от того, справедливо ли условие Q = Λ, установить P ← P + SIZE(P). (Операция NODE(Q) ← NODE(P) приводит к перемещению SIZE(P) слов; всегда Q ≤ P, так что перемещение этих слов из меньших адресов в бóльшие безопасно.) ▮

[Этот метод называется "сборщик мусора в LISP 2". Другой интересный метод, не требующий наличия поля LINK в начале узла, может основываться на идее связывания всех указателей на каждый узел. См. Lars-Erik Thorelli, *BIT* **16** (1976), 426–441; R. B. K. Dewar and A. P. McCann, *Software Practice & Exp.* **7** (1977), 95–113; F. Lockwood Morris, *CACM* **21** (1978), 662–665, **22** (1979), 571; H. B. M. Jonkers, *Inf. Proc. Letters* **9** (1979), 26–30; J. J. Martin, *CACM* **25** (1982), 571–581; F. Lockwood Morris, *Inf. Proc. Letters* **15** (1982), 139–142, **16** (1983), 215. Другие методы можно найти в работах B K. Haddon and W. M. Waite, *Comp. J.* **10** (1967), 162–165; B. Wegbreit, *Comp. J.* **15** (1972), 204–208; D. A. Zave, *Inf. Proc. Letters* **3** (1975), 167–169. Коэн (Cohen) и Николау (Nicolau) проанализировали четыре из этих подходов в *ACM Trans. Prog. Languages and Systems* **5** (1983), 532–553.]

34. Пусть TOP $\equiv$ rI1, Q $\equiv$ rI2, P $\equiv$ rI3, $k \equiv$ rI4, SIZE(P) $\equiv$ rI5. Положим также для упрощения шага G4, что $\Lambda = 0$ и LINK(0) $\neq 0$. Шаг G1 опущен.

```
01  LINK EQU  4:5
02  INFO EQU  0:3
03  SIZE EQU  1:2
04  T    EQU  3:3
05  G2   LD1  USE          1     G2. Инициализация фазы маркировки. TOP ← USE.
06       LD2  AVAIL        1
07       ST2  0,1(LINK)    1     LINK(TOP) ← AVAIL.
08       STZ  0,2(LINK)    1     LINK(AVAIL) ← Λ.
09  G3   ENT3 0,1        a+1     G3. Поднятие стека. P ← TOP.
10       LD1  0,1(LINK)   a+1     TOP ← LINK(TOP).
11       J1Z  G5          a+1     Перейти к шагу G5 при TOP = Λ.
12  G4   LD4  0,3(T)       a     G4. Поместить новые связи в стек. k ← T(P).
13  1H   J4Z  G3         a+b     k = 0?
14       INC3 1           b      P ← P + 1.
15       DEC4 1           b      k ← k − 1.
16       LD2  0,3(LINK)   b      Q ← LINK(P).
17       LDA  0,2(LINK)   b
18       JANZ 1B          b      Переход, если LINK(Q) ≠ Λ.
19       ST1  0,2(LINK)   a−1    Иначе — установить LINK(Q) ← TOP,
20       ENT1 0,2         a−1      TOP ← Q.
21       JMP  1B          a−1
22  G5   ENT2 1           1     G5. Инициализация следующей фазы. Q ← 1.
23       ST2  0,3         1     LINK(AVAIL) ← 1, SIZE(AVAIL) ← 0.
24       ENT3 1           1     P ← 1.
25       JMP  G6          1
26  1H   ST2  0,3(LINK)    a     LINK(P) ← Q.
27       INC2 0,5          a     Q ← Q + SIZE(P).
28       INC3 0,5          a     P ← P + SIZE(P).
29  G6   LDA  0,3(LINK)   a+1     G6. Присвоение новых адресов.
30  G6A  LD5  0,3(SIZE)  a+c+1
31       JAZ  G7         a+c+1   Переход, если LINK(P) = Λ.
32       J5NZ 1B          a+1    Переход, если SIZE(P) ≠ 0.
33  G8   LD1  USE          1     G8. Преобразование всех связей.
34       LDA  0,1(LINK)    1
35       STA  USE          1     USE ← LINK(USE).
36       ST2  AVAIL        1     AVAIL ← Q.
37       ENT3 1            1     P ← 1.
38       JMP  G8P          1
39  1H   LD6  0,6(SIZE)    d
40       INC5 0,6          d     rI5 ← rI5 + SIZE(P + SIZE(P)).
41  G7   ENT6 0,3         c+d    G7. Слияние свободных областей.
42       INC6 0,5          c+d   rI6 ← P + SIZE(P).
43       LDA  0,6(LINK)    c+d
44       JAZ  1B           c+d   Переход, если LINK(rI6) ≡ Λ.
45       ST5  0,3(SIZE)     c    SIZE(P) ← rI5.
46       INC3 0,5           c    P ← P + SIZE(P).
47       JMP  G6A           c
48  2H   DEC4 1            b     k ← k − 1.
```

```
49        INC2 1              b        Q ← Q + 1.
50        LD6  0,2(LINK)      b
51        LDA  0,6(LINK)      b
52        STA  0,2(LINK)      b        LINK(Q) ← LINK(LINK(Q)).
53  1H    J4NZ 2B           a + b      Переход, если k ≠ 0.
54  3H    INC3 0,5          a + c      P ← P + SIZE(P).
55  G8P   LDA  0,3(LINK)  1 + a + c
56        LD5  0,3(SIZE)  1 + a + c
57        JAZ  3B         1 + a + c    LINK(P) = Λ?
58        LD4  0,3(T)       1 + a      k ← T(P).
59        ENT2 0,3          1 + a      Q ← P.
60        J5NZ 1B           1 + a      Переход, пока не будет получено SIZE(P) = 0.
61  G9    ENT3 1              1        G9. Перемещение. P ← 1.
62        ENT1 1              1        Установить rI1 для операции MOVE.
63        JMP  G9P            1
64  1H    STZ  0,3(LINK)      a        LINK(P) ← Λ.
65        ST5  *+1(4:4)       a
66        MOVE 0,3(*)         a        NODE(rI1) ← NODE(P), rI1 ← rI1 + SIZE(P).
67  3H    INC3 0,5          a + c      P ← P + SIZE(P).
68  G9P   LDA  0,3(LINK)  1 + a + c
69        LD5  0,3(SIZE)  1 + a + c
70        JAZ  3B         1 + a + c    Переход, если LINK(P) = Λ.
71        J5NZ 1B           1 + a      Переход, пока не будет получено SIZE(P) = 0.  ▮
```

В строке 66 предполагается, что размер каждого узла достаточно мал, так что он может быть перемещен с помощью одной операции MOVE. Похоже, что это предположение применимо в большинстве случаев такой сборки мусора.

Общее время работы данной программы составляет $(44a + 17b + 2w + 25c + 8d + 47)u$, где a — количество доступных узлов, b — количество полей связи в них, c — количество недоступных узлов, которым *не предшествует* недоступный узел, d — количество недоступных узлов, которым *предшествует* недоступный узел, и w — общее количество слов в доступных узлах. Если в памяти содержится n узлов, и ρn из них недоступно, то можно оценить $a = (1 - \rho)n$, $c = (1 - \rho)\rho n$, $d = \rho^2 n$, например узлы, состоящие в среднем из пяти слов, в среднем с двумя полями связей на узел и памятью на 1000 узлов. Тогда при $\rho = \frac{1}{5}$ программа требует $374u$ времени на восстановление одного свободного узла; при $\rho = \frac{1}{2}$ время работы составляет $104u$, а при $\rho = \frac{4}{5}$ — всего $33u$.

36. Посетитель, который пришел один, может сесть на одно из 16 мест — 1, 3, 4, 6, ..., 23. Если приходит пара, для нее должно оставаться свободное место, так как иначе минимум два посетителя будут находиться на местах $(1, 2, 3)$, минимум два — на местах $(4, 5, 6)$, ..., минимум два — на местах $(19, 20, 21)$ и минимум один — на месте 22 или 23, так что в этот момент в закусочной будет находиться как минимум 15 человек.

37. Хозяйка усадила первых 16 одиноких мужчин. В результате между занятыми местами есть 17 промежутков, включая промежутки на концах ряда мест (предполагается, что между соседними занятыми местами имеется промежуток нулевой длины). Общее количество пустых мест, т. е. сумма всех семнадцати промежутков, равна 6. Предположим, что имеется x промежутков нечетной длины; тогда к услугам вновь вошедшей пары — $6 - x$ свободных мест. (Заметьте, что $6 - x$ четно и ≥ 0.) Теперь каждый посетитель номер 1, 3, 5, 7, 9, 11, 13, 15 слева направо, у которого четны промежутки с обеих сторон, встает и уходит. Каждый нечетный промежуток препятствует уходу не более одного из посетителей, следовательно, уйдут по меньшей мере $8 - x$ человек. Остается *все еще* только $6 - x$ мест для того, чтобы посадить вошедшие пары, которых вдруг оказывается $(8 - x)/2$.

38. Доказательство легко обобщается: $N(n,2) = \lfloor (3n-1)/2 \rfloor$ для $n \geq 1$. [Если хозяйка использует стратегию первого подходящего вместо оптимальной, то, как доказано Робсоном, *необходимо и достаточно* $\lfloor (5n-2)/3 \rfloor$ *мест.*]

39. Разделим память на три независимые области размерами $N(n_1, m)$, $N(n_2, m)$ и $N(2m-2, m)$. Для обработки запроса на выделение памяти поместим каждый блок в первую же область, для которой указанная емкость не будет превышена, с помощью подходящей оптимальной стратегии для этой области. Такое выделение не может не работать, так как если невозможно выполнить запрос для x ячеек памяти, то должно быть минимум $(n_1 - x + 1) + (n_2 - x + 1) + (2m - x - 1) > n_1 + n_2 - x$ уже занятых ячеек.

Теперь, если $f(n) = N(n,m) + N(2m-2, m)$, то для $f(n)$ выполняется свойство полуаддитивности $f(n_1 + n_2) \leq f(n_1) + f(n_2)$. Следовательно, существует $\lim f(n)/n$. (*Доказательство.* $f(a+bc) \leq f(a) + bf(c)$; значит,

$$\limsup_{n\to\infty} f(n)/n = \max_{0\leq a<c} \limsup_{b\to\infty} f(a+bc)/(a+bc) \leq f(c)/c$$

для всех c и

$$\limsup_{n\to\infty} f(n)/n \leq \liminf_{n\to\infty} f(n)/n. \ )$$

Таким образом, существует $\lim N(n,m)/n$.

[Из упр. 38 мы знаем, что $N(2) = \frac{3}{2}$. Значение $N(m)$ неизвестно для любого $m > 2$. Несложно показать, что множитель для двух размеров блоков, 1 и b, равен $2 - 1/b$, поэтому $N(3) \geq 1\frac{2}{3}$. Методы Робсона приводят к выводу о том, что $N(3) \leq 1\frac{11}{12}$ и $2 \leq N(4) \leq 2\frac{1}{6}$.]

40. Робсон доказал, что $N(2^r) \leq 1 + r$, используя следующую стратегию: выделить для каждого блока размером k, где $2^m \leq k < 2^{m+1}$, первый свободный блок из k ячеек, начинающийся с адреса, который кратен 2^m.

Пусть $N(\{b_1, b_2, \ldots, b_n\})$ обозначает мультипликативный коэффициент, когда все размеры блоков вынуждены находиться в множестве $\{b_1, b_2, \ldots, b_n\}$, так что $N(n) = N(\{1, 2, \ldots, n\})$. Робсон и Ш. Кроглал (S. Krogdahl) открыли, что $N(\{b_1, b_2, \ldots, b_n\}) = n - (b_1/b_2 + \cdots + b_{n-1}/b_n)$ при b_i, кратном b_{i-1}, для $1 < i \leq n$. В действительности Робсон нашел *точную* формулу $N(2^r m, \{1, 2, 4, \ldots, 2^r\}) = 2^r m(1 + \frac{1}{2}r) - 2^r + 1$. Таким образом, в частности, $N(n) \geq 1 + \frac{1}{2}\lfloor \lg n \rfloor$. Он также определил верхнюю границу $N(n) \leq 1.1825 \ln n + O(1)$ и на основе экспериментов предположил, что $N(n) = H_n$. Это предположение можно было бы легко доказать, если бы $N(\{b_1, b_2, \ldots, b_n\})$ было равно $n - (b_1/b_2 + \cdots + b_{n-1}/b_n)$, но, к сожалению, это не так, поскольку Робсон доказал, что $N(\{3,4\}) \geq 1\frac{4}{15}$. (См. *Inf. Proc. Letters* **2** (1973), 96–97; *JACM* **21** (1974), 491–499.)

41. Рассмотрим поддержку блоков размером 2^k: либо запросы на размеры 1, 2, 4, ..., 2^{k-1} будут периодически вызывать разделение нового блока размером 2^k, либо будет возвращаться блок этого размера. Можно доказать по индукции по k, что общая память, расходуемая при таком разделении блоков, никогда не превышает kn. После каждого запроса на разделение блока размером 2^{k+1} используется не более kn ячеек памяти при разделении 2^k-блоков и не более n ячеек памяти без разделения.

Доказательство можно усилить, показав, что достаточно $a_r n$ ячеек, где $a_0 = 1$ и $a_k = 1 + a_{k-1}(1 - 2^{-k})$. Имеем

$k =$	0	1	2	3	4	5
$a_k =$	1	$1\frac{1}{2}$	$2\frac{1}{8}$	$2\frac{55}{64}$	$3\frac{697}{1024}$	$4\frac{18535}{32768}$

Напротив, для $r \leq 5$ может быть показано, что система двойников иногда *требует* $a_r n$ ячеек, если механизм шагов R1 и R2 модифицирован так, чтобы выбрать наихудший из возможных 2^j-блоков вместо первого такого блока.

Доказательство Робсона того, что $N(2^r) \leq 1 + r$ (см. упр. 40), легко модифицировать, чтобы показать, что такая "самая левая" стратегия никогда не потребует более $(1 + \frac{1}{2}r)n$

ячеек для выделения пространства под блоки размерами 1, 2, 4, ..., 2^r, поскольку блоки размером 2^k никогда не будут размещаться по адресам $\geq (1+\frac{1}{2}k)n$. Хотя его алгоритм кажется очень похожим на систему двойников, он приведет к тому, что система двойников не будет такой хорошей, даже если модифицировать шаги R1 и R2, чтобы для разделения выбирались наилучшие возможные 2^j-блоки. Например, рассмотрим такую последовательность "снимков" памяти для $n = 16$ и $r = 3$.

```
11111111  11111111  00000000  00000000
10101010  10101010  2-2-2-2-  00000000
11110000  11110000  2-110000  00000000
11111111  11110000  11110000  00000000
10101010  10102-2-  10102-2-  00000000
10001000  10002-00  10002-00  4---4---
10000000  10000000  10000000  4---0000
```

Здесь 0 означает свободный адрес, а k — начало k-блока. Также имеется последовательность операций, когда n кратно 16, которая приводит к заполнению $\frac{3}{16}n$ блоков размером 8 на $\frac{1}{8}$, а других $\frac{1}{16}n$ блоков — на $\frac{1}{2}$. Если n кратно 128, последовательные запросы на $\frac{9}{128}n$ блоков размером 8 потребуют больше чем $2.5n$ ячеек памяти. (Система двойников позволяет нежелательным единицам "переползти" в $\frac{3}{16}n$ из 8-блоков, поскольку нет других свободных двоек для разделения в критический момент; алгоритм "самого левого" поддерживает все единицы ограниченными.)

42. Можно считать, что $m \geq 6$. Основная идея заключается в установлении шаблона занятости $R_{m-2}(F_{m-3}R_1)^k$ в начале памяти для $k = 0, 1, \ldots$, где R_j и F_j обозначают выделенные и свободные блоки размером j. Переход от k к $k+1$ начинается с

$$\begin{aligned} R_{m-2}(F_{m-3}R_1)^k &\to R_{m-2}(F_{m-3}R_1)^k R_{m-2}R_{m-2} \\ &\to R_{m-2}(F_{m-3}R_1)^{k-1}F_{2m-4}R_{m-2} \\ &\to R_{m-2}(F_{m-3}R_1)^{k-1}R_mR_{m-5}R_1R_{m-2} \\ &\to R_{m-2}(F_{m-3}R_1)^{k-1}F_mR_{m-5}R_1\,; \end{aligned}$$

затем коммутационная последовательность $F_{m-3}R_1F_mR_{m-5}R_1 \to F_{m-3}R_1R_{m-2}R_2R_{m-5}R_1 \to F_{2m-4}R_2R_{m-5}R_1 \to R_mR_{m-5}R_1R_2R_{m-5}R_1 \to F_mR_{m-5}R_1F_{m-3}R_1$ используется k раз до тех пор, пока не будет получено $F_mR_{m-5}R_1(F_{m-3}R_1)^k \to F_{2m-5}R_1(F_{m-3}R_1)^k \to R_{m-2}(F_{m-3}R_1)^{k+1}$. И наконец, когда k становится достаточно большим, наступает завершающая фаза, которая вызывает переполнение, если только размер памяти не превышает $(n-4m+11)(m-2)$. Детальное описание можно найти в *Comp. J.* **20** (1977), 242–244. [Заметьте, что наихудший из возможных наихудших случаев, который начинается с шаблона $F_{m-1}R_1F_{m-1}R_1F_{m-1}R_1\ldots$, только немногим хуже этого; к такому шаблону может привести стратегия "следующего подходящего", рассмотренная в упр. 6.]

43. Покажем, что если $D_1, D_2, \ldots$ представляет собой последовательность чисел, такую, что $D_1/m + D_2/(m+1) + \cdots + D_m/(2m-1) \geq 1$ для всех $m \geq 1$, и если $C_m = D_1/1 + D_2/2 + \cdots + D_m/m$, то $N_{\rm FF}(n,m) \leq nC_m$. В частности, поскольку

$$\frac{1}{m} + \frac{1}{m+1} + \cdots + \frac{1}{2m+1} = 1 - \frac{1}{2} + \cdots + \frac{1}{2m-3} - \frac{1}{2m-2} + \frac{1}{2m-1} > \ln 2,$$

последовательность констант $D_m = 1/\ln 2$ удовлетворяет необходимым условиям. Доказательство осуществляется по индукции по m. Пусть $N_j = nC_j$ для $j \geq 1$, и предположим, что некоторый запрос на блок размером m не может быть удовлетворен в результате выделения в N_m левых ячеек памяти. Тогда $m > 1$. Для $0 \leq j < m$ обозначим через N'_j крайние справа позиции, выделенные для блоков размерами $\leq j$ (либо это значение равно 0, если

все выделенные блоки больше j). По индукции имеем $N'_j \le N_j$. Кроме того, пусть N'_m означает крайние справа занятые позиции $\le N_m$, так что $N'_m \ge N_m - m + 1$. Тогда интервал $(N'_{j-1}\,..\,N'_j]$ содержит как минимум $\lceil j(N'_j - N'_{j-1})/(m+j-1)\rceil$ занятых ячеек, поскольку размеры свободных блоков в нем меньше m, а размеры его зарезервированных блоков $\ge j$. Отсюда следует, что $n - m$ не меньше числа занятых ячеек, а последнее не меньше $\sum_{j=1}^{m} j(N'_j - N'_{j-1})/(m+j-1) = mN'_m/(2m-1) - (m-1)\sum_{j=1}^{m-1} N'_j/(m+j)(m+j-1) > mN_m/(2m-1) - m - (m-1)\sum_{j=1}^{m-1} N_j(1/(m+j-1) - 1/(m+j)) = \sum_{j=1}^{m} nD_j/(m+j-1) - m \ge n - m$, что приводит к противоречию.

[Здесь доказано несколько больше, чем требовалось. Если определить D как $D_1/m + \cdots + D_m/(2m-1) = 1$, то последовательностью C_1, C_2, ... будет 1, $\frac{7}{4}$, $\frac{161}{72}$, $\frac{7483}{2880}$, ...; результат может быть улучшен даже для $m = 2$, как в упр. 38.]

44. $\lceil F^{-1}(1/N)\rceil$, $\lceil F^{-1}(2/N)\rceil$, ..., $\lceil F^{-1}(N/N)\rceil$.

ТАБЛИЦЫ ЗНАЧЕНИЙ НЕКОТОРЫХ КОНСТАНТ

Таблица 1

ВЕЛИЧИНЫ, ЧАСТО ИСПОЛЬЗУЕМЫЕ В СТАНДАРТНЫХ ПОДПРОГРАММАХ И ПРИ АНАЛИЗЕ КОМПЬЮТЕРНЫХ ПРОГРАММ (40 ДЕСЯТИЧНЫХ ЗНАКОВ)

$\sqrt{2} = 1.41421\ 35623\ 73095\ 04880\ 16887\ 24209\ 69807\ 85697-$
$\sqrt{3} = 1.73205\ 08075\ 68877\ 29352\ 74463\ 41505\ 87236\ 69428+$
$\sqrt{5} = 2.23606\ 79774\ 99789\ 69640\ 91736\ 68731\ 27623\ 54406+$
$\sqrt{10} = 3.16227\ 76601\ 68379\ 33199\ 88935\ 44432\ 71853\ 37196-$
$\sqrt[3]{2} = 1.25992\ 10498\ 94873\ 16476\ 72106\ 07278\ 22835\ 05703-$
$\sqrt[3]{3} = 1.44224\ 95703\ 07408\ 38232\ 16383\ 10780\ 10958\ 83919-$
$\sqrt[4]{2} = 1.18920\ 71150\ 02721\ 06671\ 74999\ 70560\ 47591\ 52930-$
$\ln 2 = 0.69314\ 71805\ 59945\ 30941\ 72321\ 21458\ 17656\ 80755+$
$\ln 3 = 1.09861\ 22886\ 68109\ 69139\ 52452\ 36922\ 52570\ 46475-$
$\ln 10 = 2.30258\ 50929\ 94045\ 68401\ 79914\ 54684\ 36420\ 76011+$
$1/\ln 2 = 1.44269\ 50408\ 88963\ 40735\ 99246\ 81001\ 89213\ 74266+$
$1/\ln 10 = 0.43429\ 44819\ 03251\ 82765\ 11289\ 18916\ 60508\ 22944-$
$\pi = 3.14159\ 26535\ 89793\ 23846\ 26433\ 83279\ 50288\ 41972-$
$= \pi/180 = 0.01745\ 32925\ 19943\ 29576\ 92369\ 07684\ 88612\ 71344+$
$1/\pi = 0.31830\ 98861\ 83790\ 67153\ 77675\ 26745\ 02872\ 40689+$
$\pi^2 = 9.86960\ 44010\ 89358\ 61883\ 44909\ 99876\ 15113\ 53137-$
$= \Gamma(1/2) = 1.77245\ 38509\ 05516\ 02729\ 81674\ 83341\ 14518\ 27975+$
$\Gamma(1/3) = 2.67893\ 85347\ 07747\ 63365\ 56929\ 40974\ 67764\ 41287-$
$\Gamma(2/3) = 1.35411\ 79394\ 26400\ 41694\ 52880\ 28154\ 51378\ 55193+$
$e = 2.71828\ 18284\ 59045\ 23536\ 02874\ 71352\ 66249\ 77572+$
$1/e = 0.36787\ 94411\ 71442\ 32159\ 55237\ 70161\ 46086\ 74458+$
$e^2 = 7.38905\ 60989\ 30650\ 22723\ 04274\ 60575\ 00781\ 31803+$
$\gamma = 0.57721\ 56649\ 01532\ 86060\ 65120\ 90082\ 40243\ 10422-$
$\ln \pi = 1.14472\ 98858\ 49400\ 17414\ 34273\ 51353\ 05871\ 16473-$
$\phi = 1.61803\ 39887\ 49894\ 84820\ 45868\ 34365\ 63811\ 77203+$
$e^\gamma = 1.78107\ 24179\ 90197\ 98523\ 65041\ 03107\ 17954\ 91696+$
$e^{\pi/4} = 2.19328\ 00507\ 38015\ 45655\ 97696\ 59278\ 73822\ 34616+$
$\sin 1 = 0.84147\ 09848\ 07896\ 50665\ 25023\ 21630\ 29899\ 96226-$
$\cos 1 = 0.54030\ 23058\ 68139\ 71740\ 09366\ 07442\ 97660\ 37323+$
$-\zeta'(2) = 0.93754\ 82543\ 15843\ 75370\ 25740\ 94567\ 86497\ 78979-$
$\zeta(3) = 1.20205\ 69031\ 59594\ 28539\ 97381\ 61511\ 44999\ 07650-$
$\ln \phi = 0.48121\ 18250\ 59603\ 44749\ 77589\ 13424\ 36842\ 31352-$
$1/\ln \phi = 2.07808\ 69212\ 35027\ 53760\ 13226\ 06117\ 79576\ 77422-$
$-\ln \ln 2 = 0.36651\ 29205\ 81664\ 32701\ 24391\ 58232\ 66946\ 94543-$

ВЕЛИЧИНЫ, ЧАСТО ИСПОЛЬЗУЕМЫЕ В СТАНДАРТНЫХ ПОДПРОГРАММАХ И ПРИ АНАЛИЗЕ КОМПЬЮТЕРНЫХ ПРОГРАММ (45 ВОСЬМЕРИЧНЫХ ЗНАКОВ)

Величины, расположенные слева от знака "=", заданы в десятичной системе счисления

$0.1 =$	*0.06314 63146 31463 14631 46314 63146 31463 14631 46315−*
$0.01 =$	*0.00507 53412 17270 24365 60507 53412 17270 24365 60510−*
$0.001 =$	*0.00040 61115 64570 65176 76355 44264 16254 02030 44672+*
$0.0001 =$	*0.00003 21556 13530 70414 54512 75170 33021 15002 35223−*
$0.00001 =$	*0.00000 24761 32610 70664 36041 06077 17401 56063 34417−*
$0.000001 =$	*0.00000 02061 57364 05536 66151 55323 07746 44470 26033+*
$0.0000001 =$	*0.00000 00153 27745 15274 53644 12741 72312 20354 02151+*
$0.00000001 =$	*0.00000 00012 57143 56106 04303 47374 77341 01512 63327+*
$0.000000001 =$	*0.00000 00001 04560 27640 46655 12262 71426 40124 21742+*
$0.0000000001 =$	*0.00000 00000 06676 33766 35367 55653 37265 34642 01627−*
$\sqrt{2} =$	*1.32404 74631 77167 46220 42627 66115 46725 12575 17435+*
$\sqrt{3} =$	*1.56663 65641 30231 25163 54453 50265 60361 34073 42223−*
$\sqrt{5} =$	*2.17067 36334 57722 47602 57471 63003 00563 55620 32021−*
$\sqrt{10} =$	*3.12305 40726 64555 22444 02242 57101 41466 33775 22532+*
$\sqrt[3]{2} =$	*1.20505 05746 15345 05342 10756 65334 25574 22415 03024+*
$\sqrt[3]{3} =$	*1.34233 50444 22175 73134 67363 76133 05334 31147 60121−*
$\sqrt[4]{2} =$	*1.14067 74050 61556 12455 72152 64430 60271 02755 73136+*
$\ln 2 =$	*0.54271 02775 75071 73632 57117 07316 30007 71366 53640+*
$\ln 3 =$	*1.06237 24752 55006 05227 32440 63065 25012 35574 55337+*
$\ln 10 =$	*2.23273 06735 52524 25405 56512 66542 56026 46050 50705+*
$1/\ln 2 =$	*1.34252 16624 53405 77027 35750 37766 40644 35175 04353+*
$1/\ln 10 =$	*0.33626 75425 11562 41614 52325 33525 27655 14756 06220−*
$\pi =$	*3.11037 55242 10264 30215 14230 63050 56006 70163 21122+*
$1^\circ = \pi/180 =$	*0.01073 72152 11224 72344 25603 54276 63351 22056 11544+*
$1/\pi =$	*0.24276 30155 62344 20251 23760 47257 50765 15156 70067−*
$\pi^2 =$	*11.67517 14467 62135 71322 25561 15466 30021 40654 34103−*
$\sqrt{\pi} = \Gamma(1/2) =$	*1.61337 61106 64736 65247 47035 40510 15273 34470 17762−*
$\Gamma(1/3) =$	*2.53347 35234 51013 61316 73106 47644 54653 00106 66046−*
$\Gamma(2/3) =$	*1.26523 57112 14154 74312 54572 37655 60126 23231 02452+*
$e =$	*2.55760 52130 50535 51246 52773 42542 00471 72363 61661+*
$1/e =$	*0.27426 53066 13167 46761 52726 75436 02440 52371 03355+*
$e^2 =$	*7.30714 45615 23355 33460 63507 35040 32664 25356 50217+*
$\gamma =$	*0.44742 14770 67666 06172 23215 74376 01002 51313 25521−*
$\ln \pi =$	*1.11206 40443 47503 36413 65374 52661 52410 37511 46057+*
$\phi =$	*1.47433 57156 27751 23701 27634 71401 40271 66710 15010+*
$e^\gamma =$	*1.61772 13452 61152 65761 22477 36553 53327 17554 21260+*
$e^{\pi/4} =$	*2.14275 31512 16162 52370 35530 11342 53525 44307 02171−*
$\sin 1 =$	*0.65665 24436 04414 73402 03067 23644 11612 07474 14505−*
$\cos 1 =$	*0.42450 50037 32406 42711 07022 14666 27320 70675 12321+*
$-\zeta'(2) =$	*0.74001 45144 53253 42362 42107 23350 50074 46100 27706+*
$\zeta(3) =$	*1.14735 00023 60014 20470 15613 42561 31715 10177 06614+*
$\ln \phi =$	*0.36630 26256 61213 01145 13700 41004 52264 30700 40646+*
$1/\ln \phi =$	*2.04776 60111 17144 41512 11436 16575 00355 43630 40651+*
$-\ln \ln 2 =$	*0.27351 71233 67265 63650 17401 56637 26334 31455 57005−*

Значения некоторых из приведенных в табл. 1 констант были вычислены с точностью до 40 знаков с помощью обычного калькулятора Джоном В. Ренчем (мл.) (John W. Wrench, Jr.) для первого издания этой книги. Когда в 70-х годах новое программное обеспечение позволило вычислить их на компьютере, оказалось, что значения, полученные Д. Ренчем, правильны. (В ответе к упр. 1.3.3–23 приводится значение еще одной фундаментальной константы с точностью до 40 знаков.)

Таблица 3

ЗНАЧЕНИЯ ГАРМОНИЧЕСКИХ ЧИСЕЛ, ЧИСЕЛ БЕРНУЛЛИ И ЧИСЕЛ ФИБОНАЧЧИ ДЛЯ МАЛЫХ ЗНАЧЕНИЙ n

n	H_n	B_n	F_n	n
0	0	1	0	0
1	1	−1/2	1	1
2	3/2	1/6	1	2
3	11/6	0	2	3
4	25/12	−1/30	3	4
5	137/60	0	5	5
6	49/20	1/42	8	6
7	363/140	0	13	7
8	761/280	−1/30	21	8
9	7129/2520	0	34	9
10	7381/2520	5/66	55	10
11	83711/27720	0	89	11
12	86021/27720	−691/2730	144	12
13	1145993/360360	0	233	13
14	1171733/360360	7/6	377	14
15	1195757/360360	0	610	15
16	2436559/720720	−3617/510	987	16
17	42142223/12252240	0	1597	17
18	14274301/4084080	43867/798	2584	18
19	275295799/77597520	0	4181	19
20	55835135/15519504	−174611/330	6765	20
21	18858053/5173168	0	10946	21
22	19093197/5173168	854513/138	17711	22
23	444316699/118982864	0	28657	23
24	1347822955/356948592	−236364091/2730	46368	24
25	34052522467/8923714800	0	75025	25
26	34395742267/8923714800	8553103/6	121393	26
27	312536252003/80313433200	0	196418	27
28	315404588903/80313433200	−23749461029/870	317811	28
29	9227046511387/2329089562800	0	514229	29
30	9304682830147/2329089562800	8615841276005/14322	832040	30

Пусть для любого x $H_x = \sum_{n\geq 1}\left(\frac{1}{n} - \frac{1}{n+x}\right)$. Тогда

$$
\begin{aligned}
H_{1/2} &= 2 - 2\ln 2,\\
H_{1/3} &= 3 - \tfrac{1}{2}\pi/\sqrt{3} - \tfrac{3}{2}\ln 3,\\
H_{2/3} &= \tfrac{3}{2} + \tfrac{1}{2}\pi/\sqrt{3} - \tfrac{3}{2}\ln 3,\\
H_{1/4} &= 4 - \tfrac{1}{2}\pi - 3\ln 2,\\
H_{3/4} &= \tfrac{4}{3} + \tfrac{1}{2}\pi - 3\ln 2,\\
H_{1/5} &= 5 - \tfrac{1}{2}\pi\phi^{3/2}5^{-1/4} - \tfrac{5}{4}\ln 5 - \tfrac{1}{2}\sqrt{5}\ln\phi,\\
H_{2/5} &= \tfrac{5}{2} - \tfrac{1}{2}\pi\phi^{-3/2}5^{-1/4} - \tfrac{5}{4}\ln 5 + \tfrac{1}{2}\sqrt{5}\ln\phi,\\
H_{3/5} &= \tfrac{5}{3} + \tfrac{1}{2}\pi\phi^{-3/2}5^{-1/4} - \tfrac{5}{4}\ln 5 + \tfrac{1}{2}\sqrt{5}\ln\phi,\\
H_{4/5} &= \tfrac{5}{4} + \tfrac{1}{2}\pi\phi^{3/2}5^{-1/4} - \tfrac{5}{4}\ln 5 - \tfrac{1}{2}\sqrt{5}\ln\phi,\\
H_{1/6} &= 6 - \tfrac{1}{2}\pi\sqrt{3} - 2\ln 2 - \tfrac{3}{2}\ln 3,\\
H_{5/6} &= \tfrac{6}{5} + \tfrac{1}{2}\pi\sqrt{3} - 2\ln 2 - \tfrac{3}{2}\ln 3
\end{aligned}
$$

и в общем случае, когда $0 < p < q$ (см. упр. 1.2.9–19),

$$H_{p/q} = \frac{q}{p} - \frac{\pi}{2}\cot\frac{p}{q}\pi - \ln 2q + 2\sum_{1\leq n<q/2}\cos\frac{2pn}{q}\pi\cdot\ln\sin\frac{n}{q}\pi.$$

ОСНОВНЫЕ ОБОЗНАЧЕНИЯ

Буквы в формулах, если не оговорено дополнительно, имеют следующий смысл.

j, k	Арифметическое выражение, принимающее целочисленное значение
m, n	Арифметическое выражение, принимающее неотрицательное целочисленное значение
x, y	Арифметическое выражение, принимающее действительное значение
f	Функция, принимающая действительное или комплексное значение
P	Выражение, значение которого — указатель (либо Λ, либо адреса компьютера)
S, T	Множество или мультимножество
α	Строка символов

Обозначение	Значение	Раздел
$V \leftarrow E$	Присвоить переменной V значение выражения E	1.1
$U \leftrightarrow V$	Значения переменных U и V поменять местами	1.1
A_n или $A[n]$	n-й элемент линейного множества A	1.1
A_{mn} или $A[m, n]$	Элемент, стоящий в строке m и столбце n прямоугольной таблицы (матрицы) A	1.1
NODE(P)	Узел (группа переменных, каждая из которых характеризуется именем своего поля), адресом которого является P; предполагается, что P $\neq \Lambda$	2.1
F(P)	Переменная в NODE(P) в поле с именем F	2.1
CONTENTS(P)	Содержимое слова, адрес которого — P	2.1
LOC(V)	Адрес переменной V в компьютере	2.1
P $\Leftarrow$ AVAIL	Присвоить указателю P адрес нового узла	2.2.3
AVAIL $\Leftarrow$ P	Возвратить NODE(P) на хранение; все его поля теряют наименования	2.2.3
top(S)	Узел вершины непустого стека S	2.2.1
$X \Leftarrow$ S	Взять из S в X: присвоить $X \leftarrow$ top(S); затем удалить top(S) из непустого стека S	2.2.1
S $\Leftarrow X$	Поместить X в S: вставить значение X в качестве нового входного значения в вершину стека S	2.2.1
$(B \Longrightarrow E;\ E')$	Условное выражение: означает E, если B истинно, и E', если B ложно	

Обозначение	Значение	Раздел
$[B]$	Характеристическая функция условия B: $(B \Rightarrow 1;\ 0)$	1.2.3
δ_{kj}	Символ Кронекера: $[j = k]$	1.2.3
$[z^n]\, g(z)$	Коэффициент при z^n в степенном ряду $g(z)$	1.2.9
$\sum_{R(k)} f(k)$	Сумма всех $f(k)$, таких, что значение k — целое и выполняется соотношение $R(k)$	1.2.3
$\prod_{R(k)} f(k)$	Произведение всех $f(k)$, таких, что значение k — целое и выполняется соотношение $R(k)$	1.2.3
$\min_{R(k)} f(k)$	Минимальное значение из всех $f(k)$, таких, что значение k — целое и выполняется соотношение $R(k)$	1.2.3
$\max_{R(k)} f(k)$	Максимальное значение из всех $f(k)$, таких, что значение k — целое и выполняется соотношение $R(k)$	1.2.3
$j \backslash k$	j делит k: $k \bmod j = 0$ и $j > 0$	1.2.4
$S \setminus T$	Разность множеств: $\{a \mid a$ принадлежит S и a не принадлежит $T\}$	
$\gcd(j, k)$	Наибольший общий делитель j и k: $\left(j = k = 0 \Rightarrow 0;\ \max_{d \backslash j,\, d \backslash k} d\right)$	1.1
$j \perp k$	j взаимно простое с k: $\gcd(j, k) = 1$	1.2.4
A^T	Транспонированная прямоугольная таблица (матрица) A: $A^T[j, k] = A[k, j]$	1.2.3
α^R	Левый обратный элемент к α	
x^y	x в степени y (когда x — положительное число)	1.2.2
x^k	x в степени k: $\left(k \geq 0 \Rightarrow \prod_{0 \leq j < k} x;\quad 1/x^{-k}\right)$	1.2.2
$x^{\bar{k}}$	$\Gamma(x + k)/\Gamma(x) = \left(k \geq 0 \Rightarrow \prod_{0 \leq j < k} (x + j);\quad 1/(x + k)^{\overline{-k}}\right)$	1.2.5
$x^{\underline{k}}$	$x!/(x - k)! = \left(k \geq 0 \Rightarrow \prod_{0 \leq j < k} (x - j);\quad 1/(x - k)^{\underline{-k}}\right)$	1.2.5
$n!$	n факториал: $\Gamma(n + 1) = n^{\underline{n}}$	1.2.5

Обозначение	Значение	Раздел
$\binom{x}{k}$	Биномиальный коэффициент: $(k < 0 \Rightarrow 0;\ x^{\underline{k}}/k!)$	1.2.6
$\binom{n}{n_1, n_2, \ldots, n_m}$	Полиномиальный коэффициент (определен только тогда, когда $n = n_1 + n_2 + \cdots + n_m$)	1.2.6
$\left[{n \atop m}\right]$	Число Стирлинга первого рода: $\sum_{0<k_1<k_2<\cdots<k_{n-m}<n} k_1 k_2 \ldots k_{n-m}$	1.2.6
$\left\{{n \atop m}\right\}$	Число Стирлинга второго рода: $\sum_{1\le k_1\le k_2\le\cdots\le k_{n-m}\le m} k_1 k_2 \ldots k_{n-m}$	1.2.6
$\{a \mid R(a)\}$	Множество всех a, таких, что выполняется соотношение $R(a)$	
$\{a_1, \ldots, a_n\}$	Множество или мультимножество $\{a_k \mid 1 \le k \le n\}$	
$\{x\}$	Дробная часть (используется, когда X — действительное число, а не множество): $x - \lfloor x \rfloor$	1.2.11.2
$[a\,..\,b]$	Замкнутый интервал: $\{x \mid a \le x \le b\}$	1.2.2
$(a\,..\,b)$	Открытый интервал: $\{x \mid a < x < b\}$	1.2.2
$[a\,..\,b)$	Полузамкнутый интервал: $\{x \mid a \le x < b\}$	1.2.2
$(a\,..\,b]$	Полуоткрытый интервал: $\{x \mid a < x \le b\}$	1.2.2
$\lvert S \rvert$	Число элементов множества S	
$\lvert x \rvert$	Абсолютная величина x: $(x \ge 0 \Rightarrow x;\ -x)$	
$\lvert \alpha \rvert$	Длина α	
$\lfloor x \rfloor$	Наибольшее целое число $\le x$: $\max_{k \le x} k$	1.2.4
$\lceil x \rceil$	Наименьшее целое число $\ge x$: $\min_{k \ge x} k$	1.2.4
$x \bmod y$	x по модулю y: $\left(y = 0 \Rightarrow x;\ x - y\lfloor x/y \rfloor\right)$	1.2.4
$x \equiv x'$ (по модулю y)	Сравнимость (конгруэнтность) по модулю y: $x \bmod y = x' \bmod y$	1.2.4
$O(f(n))$	О большое от $f(n)$ при $n \to \infty$	1.2.11.1
$O(f(z))$	О большое от $f(z)$ при $z \to 0$	1.2.11.1
$\Omega(f(n))$	Омега большое от $f(n)$ при $n \to \infty$	1.2.11.1
$\Theta(f(n))$	Тета большое от $f(n)$ при $n \to \infty$	1.2.11.1
$\log_b x$	Логарифм числа x по основанию b (когда $x > 0$, $b > 0$ и $b \ne 1$): y такое, что $x = b^y$	1.2.2
$\ln x$	Натуральный логарифм: $\log_e x$	1.2.2
$\lg x$	Логарифм числа x по основанию 2: $\log_2 x$	1.2.2
$\exp x$	Показательная функция от x: e^x	1.2.2

Обозначение	Значение	Раздел
$\langle X_n \rangle$	Бесконечная последовательность $X_0, X_1, X_2, \ldots$ (здесь n — часть обозначения)	1.2.9
$f'(x)$	Производная от f по x	1.2.9
$f''(x)$	Вторая производная от f по x	1.2.10
$f^{(n)}(x)$	n-я производная от f по x: $\big(n = 0 \Rightarrow f(x);\ g'(x)\big)$, где $g(x) = f^{(n-1)}(x)$	1.2.11.2
$H_n^{(x)}$	Гармоническое число порядка x: $\sum_{1 \le k \le n} 1/k^x$	1.2.7
H_n	Гармоническое число: $H_n^{(1)}$	1.2.7
F_n	Число Фибоначчи: $(n \le 1 \Rightarrow n;\ F_{n-1} + F_{n-2})$	1.2.8
B_n	Число Бернулли: $n!\,[z^n]\, z/(e^z - 1)$	1.2.11.2
$\det(A)$	Определитель квадратной матрицы A	1.2.3
$\operatorname{sign}(x)$	Знак x: $[x > 0] - [x < 0]$	
$\zeta(x)$	Дзета-функция: $\lim_{n\to\infty} H_n^{(x)}$ (где $x > 1$)	1.2.7
$\Gamma(x)$	Гамма-функция: $(x-1)! = \gamma(x, \infty)$	1.2.5
$\gamma(x, y)$	Неполная гамма-функция: $\int_0^y e^{-t} t^{x-1}\, dt$	1.2.11.3
γ	Константа Эйлера: $\lim_{n\to\infty}(H_n - \ln n)$	1.2.7
e	Основание натурального логарифма: $\sum_{n\ge 0} 1/n!$	1.2.2
π	Отношение длины окружности к ее диаметру: $4\sum_{n\ge 0}(-1)^n/(2n+1)$	
∞	Бесконечность: больше любого числа	
Λ	Пустая связь (указатель без адреса)	2.1
ϵ	Пустая строка (строка длины нуль)	
$\emptyset$	Пустое множество (множество, не содержащее элементов)	
ϕ	Золотое сечение: $\frac{1}{2}\big(1 + \sqrt{5}\big)$	1.2.8
$\varphi(n)$	Функция Эйлера: $\sum_{0 \le k < n} [k \perp n]$	1.2.4
$x \approx y$	x приближенно равно y	1.2.5, 4.2.2
$\Pr\big(S(X)\big)$	Вероятность того, что утверждение $S(X)$ справедливо для случайных величин X	1.2.10
$\mathrm{E}\, X$	Математическое ожидание (среднее значение) случайной величины X: $\sum_x x \Pr(X = x)$	1.2.10
$\operatorname{mean}(g)$	Среднее значение распределения вероятностей, заданного производящей функцией g: $g'(1)$	1.2.10
$\operatorname{var}(g)$	Дисперсия распределения вероятностей, заданного производящей функцией g: $g''(1) + g'(1) - g'(1)^2$	1.2.10

Обозначение	Значение	Раздел
(min x_1, ave x_2, max x_3, dev x_4)	Случайная величина с минимальным значением x_1, средним значением (математическим ожиданием) x_2, максимальным значением x_3, среднеквадратичным отклонением x_4	1.2.10
P*	Адрес последователя при прямом порядке обхода узла NODE(P) бинарного дерева или дерева	2.3.1, 2.3.2
P$	Адрес последователя при центрированном порядке обхода узла NODE(P) бинарного дерева, последователя дерева при обратном порядке обхода	2.3.1, 2.3.2
P♯	Адрес последователя при обратном порядке обхода узла NODE(P) бинарного дерева	2.3.1
*P	Адрес предшественника при прямом порядке обхода узла NODE(P) бинарного дерева или дерева	2.3.1, 2.3.2
$P	Адрес предшественника при центрированном порядке обхода узла NODE(P) бинарного дерева, предшественника при обратном порядке обхода дерева	2.3.1, 2.3.2
♯P	Адрес предшественника при обратном порядке обхода узла NODE(P) бинарного дерева	2.3.1
∎	Конец алгоритма, программы или доказательства	1.1
␣	Один пробел	1.3.1
rA	Регистр A (сумматор) компьютера MIX	1.3.1
rX	Регистр X (расширение) компьютера MIX	1.3.1
rI1, ..., rI6	Индексные регистры I1, ..., I6 компьютера MIX	1.3.1
rJ	Регистр перехода J компьютера MIX	1.3.1
(L:R)	Частичное поле слова компьютера MIX, $0 \le L \le R \le 5$	1.3.1
OP ADDRESS,I(F)	Обозначение команды компьютера MIX	1.3.1, 1.3.2
u	Единица времени компьютера MIX	1.3.1
*	"Сам" ("self") в языке MIXAL	1.3.2
0F, 1F, 2F, ..., 9F	"Вперед" ("forward") — локальный символ в языке MIXAL	1.3.2
0B, 1B, 2B, ..., 9B	"Назад" ("backward") — локальный символ в языке MIXAL	1.3.2
0H, 1H, 2H, ..., 9H	"Здесь" ("here") — локальный символ в языке MIXAL	1.3.2