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 упоминанин 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 вводится 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

Кулинария — это искусство, благородная наука;
все кулинары — джентльмены.
— ТИТ ЛИВИЙ, *Ab Urbe Condita* XXXIX.vi
(ROBERT BURTON, *Anatomy of Melancholy* 1.2.2.2)

НАСТОЯЩИЙ ТОМ является логическим продолжением материала об информационных структурах, содержащегося в главе 2 тома 1, так как в нем к основным структурным идеям добавляется понятие линейно упорядоченных данных.

Название "Сортировка и поиск" может ввести читателя в заблуждение, и он решит, что эта книга предназначена только для тех системных программистов, которые занимаются подготовкой программ сортировки общего назначения или приложений, выполняющих выборку информации. Но на самом деле тема сортировки и поиска представляет собой идеальную основу для обсуждения широкого круга важных общих вопросов.

- Насколько хороши разработанные алгоритмы?
- Как улучшить имеющиеся алгоритмы и программы?
- Как с помощью математических методов проанализировать эффективность алгоритмов?
- Как правильно выбрать алгоритм для решения поставленной задачи?
- В каком смысле алгоритм можно считать наилучшим из возможных?
- Какое влияние друг на друга оказывают теория и практика вычислений?
- Как эффективно использовать для больших баз данных внешние носители информации, такие как магнитные ленты, барабаны и диски?

На самом деле я считаю, что практически каждый важный аспект программирования возникает в связи с сортировкой и поиском!

В данный том входят главы 5 и 6. Глава 5 посвящена сортировке в определенном порядке. Эта большая тема разделена на две основные части, в которых речь идет о внутренней и внешней сортировке. В главу 5 включены также дополнительные разделы. В них излагаются вспомогательные теории подстановок (раздел 5.1) и оптимальных методов сортировки (раздел 5.3). В главе 6 изучается проблема поиска определенных элементов в таблицах или файлах. Здесь рассматриваются методы последовательного поиска, методы сравнения ключей либо цифровых свойств и хеширования, а также исследуется более сложная проблема — выборка вторичного ключа. Главы 5 и 6 очень взаимосвязаны между собой; многие их темы аналогичны.

В них обсуждается два различных варианта информационных структур (в дополнение к тем, которые рассматривались в главе 2), а именно: очереди с приоритетами (раздел 5.2.3) и линейные списки, представляемые в качестве сбалансированных деревьев (раздел 6.2.3).

Как и в тома 1 и 2, в этот том включен большой объем ранее не публиковавшегося материала. Многие рассказывали или писали мне о своих идеях, и я надеюсь, что я не слишком исказил их мысли, выразив их собственными словами.

У меня нет времени на систематическое изучение патентоведческой литературы. Кроме того, я осуждаю нынешнюю тенденцию к получению патентов на алгоритмы (раздел 5.4.5). Если кто-либо пришлет мне копию соответствующего патента, на который нет ссылки в данной книге, то я обязательно сделаю такую ссылку в следующих изданиях. Но все-таки я призываю современных специалистов следовать многовековой математической традиции — делать вновь найденные алгоритмы предметом всеобщего достояния. Существуют более достойные способы зарабатывания на жизнь, чем не давать другим пользоваться своим вкладом в компьютерную науку.

Еще будучи преподавателем я использовал данную книгу в качестве учебника для студентов по структурам данных. Я читал этот курс и на младших, и на старших курсах, опуская бóльшую часть математического материала. В то же время более сложные математические выкладки я использовал как основу для курса анализа алгоритмов, который читался на старших курсах университета (главным образом, это относится к разделам 5.1, 5.2.2 и 6.4). Курс по сложности вычислений (предназначенный для выпускного курса университета) может быть основан на разделах 5.3 и 5.4.4, а также 4.3.3, 4.6.3 и 4.6.4 тома 2.

Настоящий том, в основном, представляет собой полную и самостоятельную книгу; исключением являются лишь разделы, касающиеся компьютера MIX, описание которого приведено в томе 1. В приложении Б приведены использованные в данной книге математические обозначения, которые иногда отличаются от принятых в традиционной математической литературе.

Предисловие ко второму изданию

Это новое издание соответствует третьим изданиям томов 1 и 2. В них я отметил завершение разработки систем TEX и METAFONT тем, что применил их для набора книг, для которых они были предназначены.

Перейдя к электронной версии книги, я получил возможность проверить каждое слово и каждый знак пунктуации в тексте. Я старался сохранить юношеский задор оригинальных предложений и в то же время внести бóльшую зрелость суждений. Были добавлены десятки новых упражнений, а на десятки старых даны новые или улучшенные ответы. Изменения коснулись всего текста, но особенно это относится к разделам 5.1.4 (перестановки и диаграммы), 5.3 (оптимальная сортировка), 5.4.9 (сортировка на диске), 6.2.2 (энтропия), 6.4 (универсальное хеширование) и 6.5 (многомерные деревья).

Таким образом, работа над книгой *Искусство программирования* продолжается. Исследования получисленных алгоритмов продвигаются с феноменальной ско-

ростью. Именно поэтому некоторые разделы данной книги начинаются пиктограммой "В процессе построения" (это своеобразное извинение за то, что приведены не самые новые данные). Например, если бы в настоящее время я читал на последнем курсе университета курс по структурам данных, то я обязательно включил бы обсуждение случайных структур, таких как рандомизированные бинарные деревья поиска и другие. Но пока я могу только сослаться на основные статьи по этому предмету, а также объявить о написании в будущем раздела 6.2.5. Мои шкафы переполнены важными материалами, которые я планирую включить в окончательное издание тома 3; оно выйдет, вероятно, через 17 лет. Но сначала я должен закончить тома 4 и 5. Я хочу, чтобы они были опубликованы сразу же, как только будут готовы к печати.

Я чрезвычайно благодарен сотням людей, которые помогали мне собирать материал в течение последних 35 лет. Большая часть тяжелой работы по подготовке этого нового издания была выполнена Филлис Винклер (Phillis Winkler), которая перенесла текст первого издания в формат TEX, Сильвио Леви (Silvio Levy), который профессионально отредактировал текст и помог подготовить несколько десятков иллюстраций, а также Джеффри Олдхэмом (Jeffrey Oldham), который конвертировал свыше 250 оригинальных иллюстраций в формат METAPOST. Большую помощь, как всегда, оказывали и сотрудники производственного отдела издательства Addison-Wesley.

Я исправил все ошибки, которые бдительные читатели обнаружили в первом 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 обложке книги, содержится текущий список всех найденных ошибок и их исправлений, о которых мне сообщили.

Станфорд, Калифорния D. E. K.
Июль 1997

Писатель пользуется известными привилегиями, в пользе которых, надеюсь, нет никаких оснований сомневаться. Так, встретив у меня непонятное место, читатель должен предположить, что под ним кроется нечто весьма полезное и глубокомысленное.

— ДЖОНАТАН СВИФТ, *Сказка бочки, предисловие* (1704)

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

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 вместе 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 приводятся 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 обложке книги).

Остаток от деления рейтинга на 5 показывает, какой объем рутинной работы потребуется для решения данной задачи. Таким образом, для выполнения 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

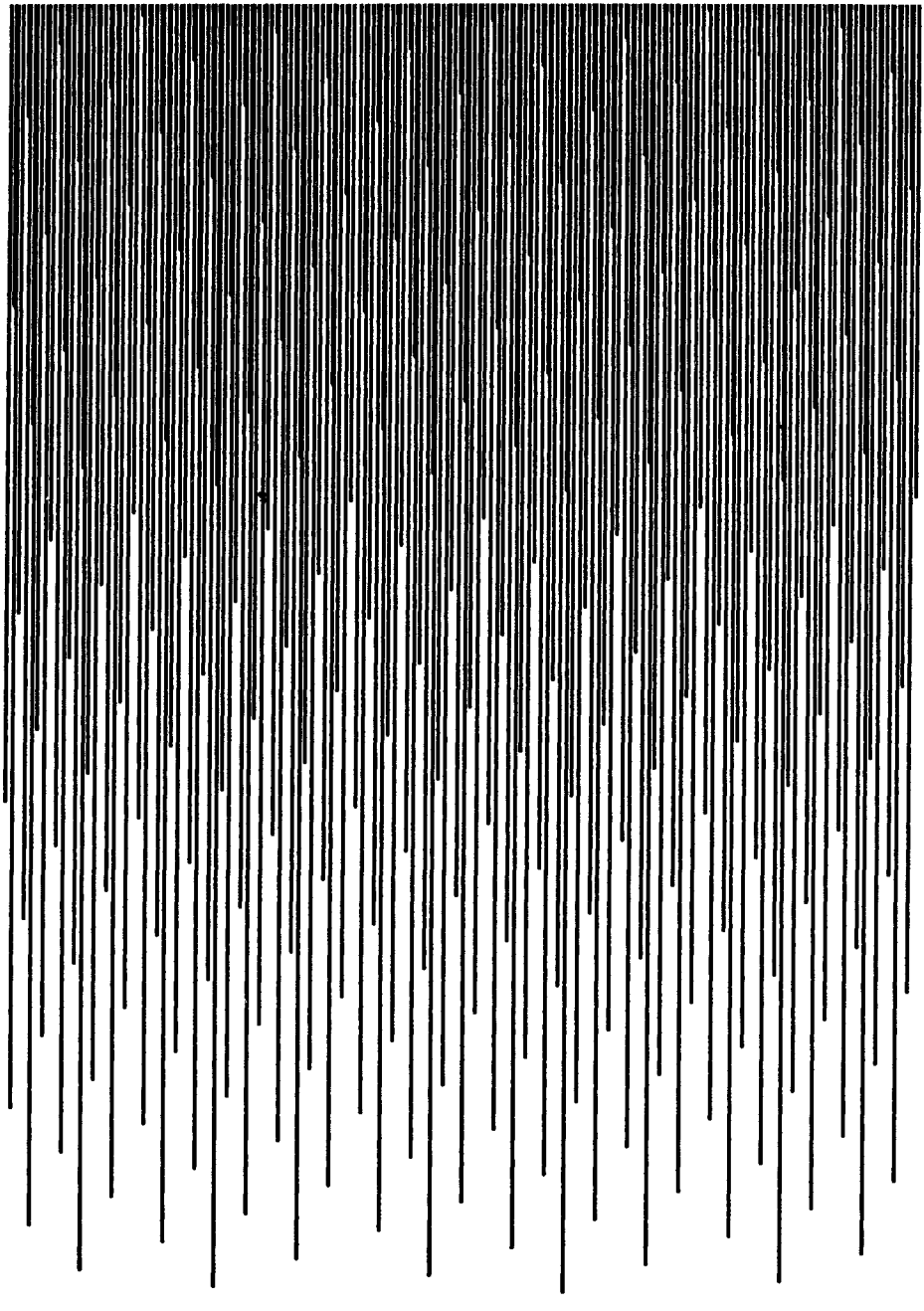
▸ **1.** [*00*] Что означает рейтинг *M20*?

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

3. [*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.

Двух часов ежедневных упражнений ... достаточно, чтобы и клячу обучить этой работе.

— М. Х. МАХОН (M. H. MAHON), *Выездка лошадей* (*The Handy Horse Book*) (1865)



ГЛАВА 5

СОРТИРОВКА

Нет дела, коего устройство было бы труднее,
ведение опаснее, а успех сомнительнее,
нежели замена старых порядков новыми.

— НИККОЛО МАКИАВЕЛЛИ, *Государь* (1513)

"Но мы не успеем просмотреть все номера автомобилей",
— возразил Дрейк.
"А нам и не нужно этого делать, Пол. Мы просто
расположим их по порядку и поищем одинаковые."

— ПЕРРИ МЕЙСОН, Из *The Case of the Angry Mourner* (1951)

Компьютер "Treesort": при новом "компьютерном подходе"
к изучению природы вы получите возможность быстро распознавать
более 260 видов деревьев США, Аляски и Канады, включая пальмы,
растительность пустынь и прочую "экзотику". Чтобы определить
породу дерева, достаточно просто вставить спицу.

— Каталог *Edmund Scientific Company* (1964)

В этой главе мы изучим вопрос, который часто возникает в программировании: переразмещение элементов в порядке возрастания или убывания. Представьте, насколько трудно было бы пользоваться словарем, если бы слова в нем не располагались в алфавитном порядке. Точно так же от порядка, в котором хранятся элементы в памяти компьютера, во многом зависит скорость выполнения и простота алгоритмов, предназначенных для их обработки.

Хотя в словарях слово "сортировка" (sorting) определяется как процесс разделения объектов по виду или сорту, программисты традиционно используют его в гораздо более узком смысле, обозначая им такую перестановку предметов, при которой они располагаются в порядке возрастания или убывания. Такой процесс, пожалуй, следовало бы назвать не сортировкой, а *упорядочением* (ordering), но использование этого слова привело бы к путанице из-за перегруженности значениями слова "порядок". Рассмотрим, например, следующее предложение: "Так как только два из имеющихся у нас лентопротяжных механизмов в порядке, меня призвали к порядку и обязали в срочном порядке заказать еще несколько устройств, чтобы можно было упорядочивать данные разного порядка на несколько порядков быстрее". В математической терминологии это слово также изобилует значениями (порядок группы, порядок перестановки, порядок точки ветвления, отношение порядка и т. п.). Итак, слово "порядок" приводит к хаосу.

В качестве обозначения для процесса упорядочения предлагалось также слово "ранжирование" (sequencing), но оно во многих случаях, по-видимому, не вполне отражает суть дела, особенно если присутствуют равные элементы, и, кроме того, иногда не согласуется с другими терминами. Конечно, слово "сортировка" и само имеет довольно много значений, но оно прочно вошло в программистский жаргон. Поэтому мы без дальнейших извинений будем использовать слово "сортировка" в узком смысле: "размещение по порядку". Вот некоторые из наиболее важных областей применения сортировки.

a) *Решение задачи группирования*, когда нужно собрать вместе все элементы с одинаковыми значениями некоторого признака. Допустим, имеется 10 000 элементов, расположенных в случайном порядке, причем значения многих из них равны и нужно переупорядочить массив так, чтобы элементы с равными значениями занимали соседние позиции в массиве. Это, по существу, тоже задача "сортировки", но в более широком смысле, и она легко может быть решена путем сортировки массива в указанном выше узком смысле, а именно — в результате расположения элементов в порядке неубывания $v_1 \le v_2 \le \cdots \le v_{10000}$. Эффект, который может быть достигнут после выполнения этой процедуры, и объясняет изменение первоначального смысла слова "сортировка".

b) *Поиск общих элементов в двух или более массивах.* Если два или более массивов рассортировать в одном и том же порядке, то можно отыскать в них все общие элементы за один последовательный просмотр всех массивов без возвратов. Именно этим принципом и воспользовался Перри Мейсон, чтобы раскрыть дело об убийстве (см. эпиграфы к этой главе). Оказывается, что, как правило, гораздо удобнее просматривать список последовательно, а не перескакивая с места на место случайным образом, если только список не настолько мал, что он целиком помещается в оперативной памяти. Сортировка позволяет использовать последовательный доступ к большим массивам в качестве приемлемой замены прямой адресации.

c) *Поиск информации по значениям ключей.* Как мы узнаем из главы 6, сортировка используется и при поиске; с ее помощью можно сделать результаты обработки данных более удобными для восприятия человеком. В самом деле, подготовленный компьютером список, рассортированный в алфавитном порядке, зачастую выглядит весьма внушительно, даже если содержащиеся в нем числовые данные были рассчитаны неверно.

Хотя сортировка традиционно и большей частью использовалась для обработки коммерческих данных, в действительности она является инструментом, полезным в самых разных ситуациях, и поэтому о ней не следует забывать. В упр. 2.3.2–17 мы обсудили ее применение для упрощения алгебраических формул. Упражнения, приведенные ниже, иллюстрируют разнообразие типичных применений сортировки.

Одной из первых мощных программных систем, продемонстрировавших богатые возможности сортировки, был компилятор LARC Scientific Compiler, разработанный Дж. Эрдвином (J. Erdwinn) и Д. Е. Фергюсоном (D. E. Ferguson) в фирме Computer Sciences Corporation в 1960 году. В этом оптимизирующем компиляторе для расширенного языка FORTRAN сортировка использовалась весьма интенсивно, так что различные алгоритмы компиляции работали с относящимися к ним фрагментами исходного текста программы, расположенными в удобной последова-

тельности. На первом проходе осуществлялся лексический анализ, т. е. выделение в исходной программе лексических единиц (лексем), каждая из которых соответствует либо идентификатору (имени переменной), либо константе, либо оператору и т. д. Каждая лексема получала несколько порядковых номеров. В результате сортировки по именам и соответствующим порядковым номерам все фрагменты текста, в которых использовался данный идентификатор, оказывались собранными вместе. "Определяющие вхождения", специфицирующие идентификатор как имя функции, простую (параметр) или многомерную (массив) переменную, получали меньшие номера, поэтому они оказывались первыми в последовательности лексем, которые соответствовали этому идентификатору. Тем самым облегчалась проверка правильности употребления идентификаторов, распределение памяти с учетом деклараций `EQUIVALENCE` и т. д. Собранная таким образом информация о каждом идентификаторе присоединялась к соответствующей лексеме. Поэтому не было необходимости хранить в оперативной памяти "таблицу символов" содержащую сведения об идентификаторах. После такой обработки лексемы снова сортировались по другому порядковому номеру; в результате в программе, по существу, восстанавливался первоначальный порядок, если не считать того, что арифметические выражения оказывались записанными в более удобной "польской префиксной" форме. Сортировка использовалась и на последующих фазах компиляции — для упрощения оптимизации циклов, включения в листинг сообщений об ошибках и т. д. Короче говоря, компилятор был спроектирован так, что всю обработку файлов (а для их хранения использовались весьма распространенные в те времена устройства внешней памяти на магнитных барабанах) можно было выполнять последовательно. Поэтому-то данные и снабжались такими порядковыми номерами, которые позволяли упорядочивать эти данные различными удобными способами.

По оценкам производителей компьютеров в 60-х годах в среднем более четверти машинного времени тратилось на сортировку. Во многих вычислительных системах на нее уходит больше половины машинного времени. Исходя из этих статистических данных можно заключить, что либо (i) сортировка имеет много важных применений, либо (ii) ею часто пользуются без нужды, либо (iii) применяются в основном неэффективные алгоритмы сортировки. По-видимому, каждое из приведенных предположений содержит долю истины.

Во всяком случае ясно, что сортировка заслуживает серьезного изучения с точки зрения ее практического использования. Но даже если бы сортировка была почти бесполезна, нашлась бы масса других причин заняться ею! Появление изощренных алгоритмов сортировки говорит о том, что она и сама по себе интересна как объект исследования. В этой области существует множество увлекательных нерешенных задач наряду с весьма немногими уже решенными.

Рассматривая вопрос в более широком плане, мы обнаружим, что алгоритмы сортировки представляют собой интересный *частный пример* того, как следует подходить к решению проблем программирования вообще. Мы познакомимся со многими важными принципами манипулирования структурами данных и проследим за эволюцией различных методов сортировки, причем читателю часто будет предоставлена возможность самостоятельно "изобрести велосипед", как будто бы до него никто с подобными задачами не сталкивался. Обобщение этих частных методов позволит нам в значительной степени овладеть теми подходами, которые помогут

создавать качественные алгоритмы для решения других проблем, связанных с использованием компьютеров.

Методы сортировки служат великолепной иллюстрацией базовых концепций *анализа алгоритмов*, т. е. оценки качества алгоритмов, что, в свою очередь, позволяет разумно делать выбор среди, казалось бы, равноценных методов. Читатели, имеющие склонность к математике, найдут в этой главе немало поучительных способов оценки скорости работы алгоритмов и методов решения сложных рекуррентных соотношений. С другой стороны, изложение построено так, что читатели, не имеющие такой склонности, могут безболезненно пропускать выкладки.

Прежде чем двигаться дальше, необходимо более четко сформулировать задачу и ввести соответствующую терминологию. Пусть надо упорядочить N элементов

$$R_1, R_2, \ldots, R_N.$$

Назовем их *записями*, а всю совокупность N записей назовем *файлом*. Каждая запись R_j имеет *ключ*, K_j, который и управляет процессом сортировки. Помимо ключа, запись может содержать дополнительную "сопутствующую информацию", которая не влияет на сортировку, но всегда остается в этой записи.

Отношение порядка "$<$" на множестве ключей вводится таким образом, чтобы для любых трех значений ключей a, b, c выполнялись следующие условия:

i) справедливо одно и только одно из соотношений $a < b$, $a = b$, $b < a$ (закон трихотомии);

ii) если $a < b$ и $b < c$, то $a < c$ (закон транзитивности).

Эти два свойства определяют математическое понятие *линейного упорядочения*, называемого также *совершенным упорядочением*. Любое множество с отношением "$<$", удовлетворяющим обоим этим свойствам, поддается сортировке большинством методов, описанных в этой главе, хотя некоторые из методов годятся только для числовых и буквенных ключей с общепринятым отношением порядка.

Задача сортировки — найти такую перестановку записей $p(1)\,p(2)\ldots p(N)$ с индексами $\{1, 2, \ldots, N\}$, после которой ключи расположились бы в порядке неубывания:

$$K_{p(1)} \le K_{p(2)} \le \cdots \le K_{p(N)}. \tag{1}$$

Сортировка называется *устойчивой*, если она удовлетворяет такому дополнительному условию, что записи с одинаковыми ключами остаются в прежнем порядке, т. е., другими словами,

$$p(i) < p(j) \qquad \text{для любых} \quad K_{p(i)} = K_{p(j)} \quad \text{и} \quad i < j. \tag{2}$$

В одних случаях придется физически перемещать записи в памяти так, чтобы их ключи были упорядочены; в других случаях достаточно создать вспомогательную таблицу, которая некоторым образом описывает перестановку и обеспечивает доступ к записям в соответствии с порядком их ключей.

Некоторые методы сортировки предполагают существование величин "∞" и "$-\infty$" или одной из них. Величина "∞" считается больше, а величина "$-\infty$" меньше любого ключа:

$$-\infty < K_j < \infty \qquad \text{для } 1 \le j \le N. \tag{3}$$

Эти величины используются в качестве искусственных ключей, а также граничных признаков. Равенство в (3), вообще говоря, исключено. Если же оно, тем не менее, допускается, алгоритмы можно модифицировать так, чтобы они все-таки работали, хотя нередко при этом их изящество и эффективность отчасти утрачиваются.

Обычно методы сортировки подразделяют на два класса: внутренние, когда все записи хранятся в быстрой оперативной памяти, и внешние, когда все записи в ней не помещаются. Методы внутренней сортировки обеспечивают большую гибкость при построении структур данных и доступа к ним, внешние же методы обеспечивают достижение нужного результата в "спартанских" условиях ограниченных ресурсов.

Достаточно хороший общий алгоритм затрачивает на сортировку N записей время пропорционально $N \log N$; при этом требуется около $\log N$ "проходов" по данным. Как мы увидим в разделе 5.3.1, это минимальное время, если записи расположены в произвольном порядке и сортировка выполняется попарным сравнением ключей. Так, если удвоить число записей, то и время при прочих равных условиях возрастет немногим более чем вдвое. (На самом деле, когда N неограниченно возрастает, время сортировки растет как $N(\log N)^2$, если все ключи различны, поскольку и размеры ключей увеличиваются, как минимум, пропорционально $\log N$ с ростом N; но практически N всегда остается ограниченным.)

С другой стороны, если известно, что ключи являются случайными величинами с некоторым непрерывным распределением, то, как мы увидим ниже, сортировка может быть выполнена в среднем за $O(N)$ шагов.

УПРАЖНЕНИЯ (часть 1)

1. [*M20*] Докажите, что из законов трихотомии и транзитивности вытекает *единственность* перестановки $p(1)\,p(2)\ldots p(N)$, если сортировка устойчива.

2. [*21*] Пусть каждая запись R_j некоторого файла имеет *два* ключа: "большой ключ" K_j и "малый ключ" k_j, причем оба множества ключей линейно упорядочены. Тогда можно обычным способом ввести *лексикографический порядок* на множестве пар ключей (K, k):

$$(K_i, k_i) < (K_j, k_j), \quad \text{если} \quad K_i < K_j \quad \text{или} \quad K_i = K_j \quad \text{и} \quad k_i < k_j.$$

Алиса рассортировала этот файл сначала по большим ключам, получив n групп записей с одинаковыми большими ключами в каждой группе:

$$K_{p(1)} = \cdots = K_{p(i_1)} < K_{p(i_1+1)} = \cdots = K_{p(i_2)} < \cdots < K_{p(i_{n-1}+1)} = \cdots = K_{p(i_n)},$$

где $i_n = N$. Затем она рассортировала каждую из n групп $R_{p(i_{j-1}+1)}, \ldots, R_{p(i_j)}$ по собственным малым ключам.

Билл взял тот же исходный файл и сначала рассортировал его по малым ключам, а затем получившийся файл рассортировал по большим ключам.

Взяв тот же исходный файл, Крис рассортировал его один раз в лексикографическом порядке, пользуясь парами ключей (K_j, k_j).

Получилось ли у всех троих одно и то же?

3. [*M25*] Пусть на множестве $K_1, \ldots, K_N$ определено отношение "<", для которого закон трихотомии выполняется, а закон транзитивности — нет. Докажите, что и в этом случае возможна устойчивая сортировка записей, т. е. сортировка, при которой выполняются условия (1) и (2). (На самом деле существует, по крайней мере, три расположения записей, удовлетворяющих этим условиям!)

▶ **4.** [*21*] Составители словарей фактически не следуют жестко лексикографическому порядку, так как прописные и строчные буквы у них перемежаются. В частности, используется такой порядок:

$$\text{a} < \text{A|aa} < \text{AA} < \text{AAA} < \text{Aachen} < \text{aah} < \cdots < \text{zzz} < \text{ZZZ}.$$

Поясните, как реализовать подобный словарный порядок сортировки.

▶ **5.** [*M28*] Разработайте двоичный код для всех неотрицательных целых чисел, такой, что, если n закодировано в виде строки $\rho(n)$, то $m < n$ тогда и только тогда, когда $\rho(m)$ меньше в лексикографическом смысле, чем $\rho(n)$. Более того, $\rho(m)$ не должно быть префиксом $\rho(n)$ для любого $m \neq n$. По возможности длина $\rho(n)$ должна быть по крайней мере $\lg n + O(\log\log n)$ для всех больших n. (Такой способ кодирования очень удобен, если приходится сортировать текст, состоящий из чисел и слов, или если желательно отобразить довольно большие текстовые последовательности на двоичные коды.)

6. [*15*] Программисту на MIX Б. С. Даллу потребовалось выяснить, находится ли в ячейке A число, большее числа из ячейки B, меньшее или же равное ему. Он написал в программе "LDA A: SUB B", а потом проверил, какой результат получился в регистре A: положительный, отрицательный или нуль. Какую серьезную ошибку он допустил и как должен был поступить?

7. [*17*] Напишите подпрограмму для компьютера MIX для сравнения ключей с увеличенной точностью, исходя из следующих условий.

Вызов:	JMP COMPARE
Состояние при входе:	rI1 $= n$; CONTENTS(A $+ k$) $= a_k$ и CONTENTS(B $+ k$) $= b_k$ для $1 \le k \le n$; полагается, что $n \ge 1$.
Состояние при выходе:	CI = GREATER, если $(a_n, \ldots, a_1) > (b_n, \ldots, b_1)$; CI = EQUAL, если $(a_n, \ldots, a_1) = (b_n, \ldots, b_1)$; CI = LESS, если $(a_n, \ldots, a_1) < (b_n, \ldots, b_1)$; rX и rI1, возможно, изменились.

Здесь отношение $(a_n, \ldots, a_1) < (b_n, \ldots, b_1)$ обозначает лексикографическое упорядочение слева направо, т. е. существует индекс j, такой, что $a_k = b_k$ для $n \ge k > j$, но $a_j < b_j$.

▶ **8.** [*30*] В ячейках A и B содержатся соответственно числа a и b. Покажите, что можно написать программу для MIX, которая вычисляла бы $\min(a, b)$ и записывала результат в ячейку C, не пользуясь командами перехода. (*Предостережение.* Поскольку арифметическое переполнение невозможно обнаружить без операторов перехода, разумно так построить программу, чтобы переполнение не могло возникнуть ни при каких значениях a и b.)

9. [*M27*] Какова вероятность того, что после сортировки в порядке неубывания n независимых равномерно распределенных на отрезке [0, 1] случайных величин r-е от начала число окажется $\le x$?

УПРАЖНЕНИЯ (часть 2)

В каждом из этих упражнений поставлена задача, с которой может столкнуться программист. Предложите хорошее решение задачи, *предполагая, что вы имеете дело с "музейным" компьютером, у которого имеется сравнительно небольшая оперативная память, порядка нескольких тысяч слов* и около полудюжины накопителей на магнитной ленте (НМЛ) — этого количества вполне достаточно для выполнения сортировки. Алгоритм, который сможет эффективно работать в таких "спартанских" условиях, тем более будет прекрасно выполнять задачу на современных компьютерах.

10. [*15*] Имеется лента, на которой записан миллион слов данных. Как определить, сколько на этой ленте различных слов?

11. [*18*] Вообразите себя в роли Управления внутренних доходов Министерства финансов США. Вы получаете миллионы информационных карточек от организаций о том, сколько денег они выплатили различным лицам, и миллионы налоговых деклараций (карточек) о доходах от налогоплательщиков. Как бы вы стали отыскивать людей, которые сообщили не обо всех своих доходах?

12. [*M25*] (*Транспонирование матрицы.*) Имеется магнитная лента, содержащая миллион слов, которые представляют собой элементы матрицы 1000×1000, записанные по строкам: $a_{1,1}\, a_{1,2} \ldots a_{1,1000}\, a_{2,1} \ldots a_{2,1000} \ldots a_{1000,1000}$. Ваша задача — получить ленту, на которой элементы этой матрицы были бы записаны по столбцам: $a_{1,1}\, a_{2,1} \ldots a_{1000,1}\, a_{1,2} \ldots a_{1000,2} \ldots a_{1000,1000}$. (Постарайтесь сделать не более десяти просмотров данных.)

13. [*M26*] В вашем распоряжении есть довольно большой файл из N слов. Как бы вы его "перетасовали" случайным образом?

14. [*20*] Вы работаете с двумя вычислительными системами, в которых по-разному упорядочены литеры (буквы и цифры). Как заставить первую систему сортировать файлы с буквенно-цифровой информацией, предназначенные для использования во второй системе?

15. [*18*] Имеется довольно большой список людей, родившихся в США, с указанием штата, в котором они родились. Как подсчитать тех, кто родился в каждом штате? (Предполагается, что ни один человек не указан в списке более одного раза.)

16. [*20*] Чтобы облегчить внесение изменений в большие программы, написанные на языке FORTRAN, желательно написать программу, которая формирует таблицу *перекрестных ссылок*. Входными данными для нее служит программа на FORTRAN, а в результате получается листинг исходной программы, снабженный указателем всех случаев употребления каждого идентификатора (т. е. имени) в программе. Как написать такую программу?

▶ **17.** [*33*] (*Сортировка библиографических карточек каталога.*) До появления компьютерных баз данных в каждой библиотеке существовал каталог библиографических карточек, с помощью которого читатели могли отыскать интересующие их издания. Но сортировка карточек в порядке, удобном для читателей, усложняется по мере увеличения библиотечного фонда. В следующем "алфавитном" списке содержатся рекомендации, взятые из правил регистрации и хранения каталожных карточек Американской библиотечной ассоциации (Чикаго, 1942).

Текст карточки	*Замечания*
R. Accademia nazionale dei Lincei, Rome	В названиях иностранных (кроме британских) учреждений слово "royalty" (королевский) игнорируется
1812; ein historischer Roman	Achtzehnhundertzwölf (числительное 1812 на немецком языке)
Bibliothèque d'histoire révolutionnaire	В французском тексте апостроф рассматривается как пробел
Bibliothèque des curiosités	Диатонические знаки игнорируются
Brown, Mrs. J. Crosby	Указание положения (Mrs.) игнорируется
Brown, John	Фамилии с датами следуют за фамилиями без дат
Brown, John, mathematician	..., которые упорядочиваются
Brown, John, of Boston	по описательным словам
Brown, John, 1715–1766	Одинаковые фамилии упорядочиваются по датам рождения
BROWN, JOHN, 1715–1766	Работы о нем идут после его работ
Brown, John, d. 1811	Иногда год рождения определяется приблизительно

Текст карточки	*Замечания*
Brown, Dr. John, 1810–1882	Указание положения (Dr.) игнорируется
Brown-Williams, Reginald Makepeace	Дефис рассматривается как пробел
Brown America	Названия книг указываются после составных фамилий
Brown & Dallison's Nevada directory	& в английском тексте превращается в "and"
Brownjohn, Alan	
Den', Vladimir Éduardovich, 1867–	Апостроф в именах игнорируется
The den	Артикль в начале текста игнорируется
Den lieben langen Tag	..., если существительное стоит в именительном падеже
Dix, Morgan, 1827–1908	Фамилии идут раньше других слов
1812 ouverture	Dix-huit cent douze (числительное 1812 на французском языке)
Le XIXe siècle français	Dix-neuvième (числительное XIX-й на французском языке)
The 1847 issue of U. S. stamps	Eighteen forty-seven (числительное 1847 на английском языке)
1812 overture	Eighteen twelve (числительное 1812 на английском языке)
I am a mathematician	(a book by Norbert Wiener) (книга Норберта Винера)
IBM journal of research and development	Аббревиатуры рассматриваются как ряд однобуквенных слов
ha-I ha-ehad	Артикль в начале текста игнорируется
Ia; a love story	Знаки препинания в названиях игнорируются
International Business Machines Corporation	
al-Khuwārizmī, Muḥammad ibn Mūsā, *fl.* 813–846	Начальное "al-" в арабских именах игнорируется
Labour. A magazine for all workers	Заменяется словом "Labor"
Labor research association	
Labour, *see* Labor	Ссылка на другую карточку в картотеке
McCall's cookbook	Апостроф в английском тексте игнорируется
McCarthy, John, 1927–	Mc = Mac
Machine-independent computer programming.	Дефис рассматривается как пробел
MacMahon, Maj. Percy Alexander, 1854–1929	Указание положения (Maj.) игнорируется
Mrs. Dalloway	"Mrs." заменяется словом "Mistress"
Mistress of mistresses	
Royal society of London	В названиях британских учреждений слово "royalty" (королевский) учитывается
St. Petersburger Zeitung	"St." заменяется словом "Saint", даже в немецком тексте
Saint-Saëns, Camille, 1835–1921	Дефис рассматривается как пробел
Ste-Marie, Gaston P	Sainte
Seminumerical algorithms	(a book by Donald Ervin Knuth)
Uncle Tom's cabin	(a book by Harriet Beecher Stowe)
U. S. bureau of the census	"U. S." заменяется словами "United States"

Текст карточки	*Замечания*
Vandermonde, Alexandre Théophile, 1735–1796	
Van Valkenburg, Mac Elwyn, 1921–	Пробел после префикса в фамилиях игнорируется
Von Neumann, John, 1903–1957	
The whole art of legerdemain	Артикль в начале текста игнорируется
Who's afraid of Virginia Woolf?	Апостроф в текстах на английском языке игнорируется
Wijngaarden, Adriaan van, 1916–	Фамилия никогда не начинается со строчной литеры

(У большинства из этих правил есть исключения; кроме того, существует много других правил, которые здесь не упомянуты.)

Предположим, вам поручено рассортировать большое количество таких карточек с помощью компьютера, а затем обслуживать огромную картотеку, причем у вас нет возможности изменить уже сложившийся порядок заполнения карточек. Как бы вы организовали информацию, чтобы упростить операции сортировки и включения новых карточек?

18. [*M25*] (Э. Т. Паркер (E. T. Parker).) Леонард Эйлер высказал предположение [*Nova Acta Acad. Sci. Petropolitanæ* **13** (1795), 45–63, §3; написана в 1778], что уравнение

$$u^6 + v^6 + w^6 + x^6 + y^6 = z^6$$

не имеет нетривиальных решений среди целых неотрицательных чисел u, v, w, x, y, z. Одновременно он предположил, что уравнение

$$x_1^n + \cdots + x_{n-1}^n = x_n^n$$

не имеет решения в положительных целых числах при $n \geq 3$, но это предположение было опровергнуто уже в наше время, когда с помощью компьютера было найдено тождество $27^5 + 84^5 + 110^5 + 133^5 = 144^5$ [см. L. J. Lander, T. R. Parkin, and J. L. Selfridge, *Math. Comp.* **21** (1967), 446–459]. Множество аналогичных примеров было обнаружено и при $n = 4$ Н. Д. Элкисом (N. D. Elkies) [*Math. Comp.* **51** (1988), 825–835]. Подумайте, как можно было бы использовать сортировку для поиска примеров, опровергающих предположение Эйлера при $n = 6$.

▶ **19.** [*24*] Файл содержит большое количество 30-разрядных двоичных слов: $x_1, \ldots, x_N$. Придумайте хороший способ нахождения в нем всех *пар с взаимно дополняющими элементами* $\{x_i, x_j\}$. (Два слова называются взаимно дополняющими, если второе содержит нули во всех разрядах, в которых были единицы в первом слове, и наоборот; таким образом, они взаимно дополняющие тогда и только тогда, когда их сумма равна $(11\ldots1)_2$, если они рассматриваются как двоичные числа.)

▶ **20.** [*25*] Имеется файл, содержащий тысячу 30-разрядных двоичных слов $x_1, \ldots, x_{1000}$. Как бы вы стали составлять список всех пар (x_i, x_j), таких, что x_i отличается от x_j не более чем в двух разрядах?

21. [*22*] Как бы вы поступили, если бы вам нужно было найти все пятибуквенные анаграммы, такие как **CARET**, **CARTE**, **CATER**, **CRATE**, **REACT**, **RECTA**, **TRACE**; **CRUEL**, **LUCRE**, **ULCER**; **DOWRY**, **ROWDY**, **WORDY**? [Или если бы вы, допустим, захотели узнать, существуют ли в английском языке наборы из десяти или более анаграмм, кроме замечательной серии

APERS, **ASPER**, **PARES**, **PARSE**, **PEARS**, **PRASE**, **PRESA**, **RAPES**, **REAPS**, **SPAER**, **SPARE**, **SPEAR**,

к которой можно еще добавить французское слово **APRÈS**.]

22. [*M28*] Пусть даны описания весьма большого числа ориентированных графов. Как можно сгруппировать *изоморфные* графы? (Два ориентированных графа называются изоморфными, если существует взаимно однозначное соответствие между их вершинами и взаимно однозначное соответствие между их дугами, причем эти соответствия сохраняют инцидентность вершин и дуг.)

23. [*30*] В некоторой группе из 4 096 человек каждый имеет около 100 знакомых. Был составлен список всех пар людей, знакомых друг с другом. (Это отношение симметрично, т. е. если x знаком с y, то и y знаком с x. Поэтому в списке содержится примерно 200 000 пар.) Придумайте алгоритм, который по заданному k выдавал бы все *клики* из k человек. (*Клика* — это группа людей, в которой все знают друг друга.) Предполагается, что не бывает слишком больших клик (размером более 25).

▶ **24.** [*30*] Три миллиона человек с различными именами были уложены рядом непрерывной цепочкой от Нью-Йорка до Калифорнии. Каждому из них дали листок бумаги, на котором он написал свое имя и имя своего ближайшего западного соседа. Человек, находившийся в крайней западной точке цепи, не понял, что ему делать, и выбросил свой листок. Остальные 2 999 999 листков собрали в большую корзину и отправили в Национальный архив, в Вашингтон, округ Колумбия. Там содержимое корзины тщательно перетасовали и записали на магнитные ленты.

Специалист по теории информации определил, что имеется достаточно сведений для восстановления списка людей в исходном порядке. Программист нашел способ сделать это менее чем за 1 000 просмотров лент с данными, используя лишь последовательный доступ к файлам на лентах и небольшой объем оперативной памяти. Как ему это удалось?

[Другими словами, как, имея расположенные произвольным образом пары (x_i, x_{i+1}), $1 \le i < N$, где все x_i различны, получить последовательность $x_1 x_2 \ldots x_N$, применяя лишь методы последовательной обработки данных, которые пригодны для работы с магнитными лентами? Это сортировка в случае, когда трудно определить, какой из двух ключей предшествует другому; мы уже поднимали этот вопрос в упр. 2.2.3–25.]

25. [*M21*] (*Дискретные логарифмы.*) Пусть известно, что p — простое число (довольно большое), а a — первообразный корень по модулю p. Следовательно, для любого b в диапазоне $1 \le b < p$ существует единственное n, такое, что $a^n \bmod p = b$, $1 \le n < p$. (Это n называется индексом b по модулю p по отношению к a.) Как по заданному b найти n менее чем за $\Omega(n)$ шагов. [*Указание.* Пусть $m = \lceil \sqrt{p} \rceil$. Попытайтесь решить уравнение $a^{mn_1} \equiv ba^{-n_2}$ (по модулю p) при $0 \le n_1, n_2 < m$.]

*5.1. КОМБИНАТОРНЫЕ СВОЙСТВА ПЕРЕСТАНОВОК

Перестановкой конечного множества называется некоторое расположение его элементов в ряд. Перестановки особенно важны при изучении алгоритмов сортировки, так как они служат для представления неупорядоченных исходных данных. Чтобы исследовать эффективность различных методов сортировки, нужно уметь подсчитывать количество перестановок, которые вынуждают повторять некоторый шаг алгоритма определенное число раз.

Мы уже не раз встречались с перестановками в предыдущих главах. Например, в разделе 1.2.5 обсуждались два основных теоретических метода построения $n!$ перестановок n объектов; в разделе 1.3.3 были проанализированы некоторые алгоритмы, связанные с циклической структурой и мультипликативными свойствами перестановок; в разделе 3.3.2 изучались их серии монотонности. Цель настоящего параграфа — описать другие свойства перестановок и рассмотреть общий случай, когда допускается наличие одинаковых элементов. Попутно мы многое узнаем о комбинаторной математике.

Свойства перестановок настолько красивы, что представляют и самостоятельный интерес. Удобно будет дать их систематическое изложение в одном месте, а не разбрасывать материал по всей главе. Читателям, не имеющим склонности к математике, а также тем, кто жаждет поскорее добраться до самих методов сортировки, рекомендуется перейти сразу к разделу 5.2, потому что настоящий раздел *непосредственного* отношения к сортировке почти не имеет.

*5.1.1. Инверсии

Пусть $a_1\,a_2\ldots a_n$ — перестановка множества $\{1,2,\ldots,n\}$. Если $i<j$ и $a_i>a_j$, то пара (a_i,a_j) называется *инверсией* перестановки; например, перестановка 3 1 4 2 имеет три инверсии: (3,1), (3,2) и (4,2). Каждая инверсия — это пара элементов, "нарушающих порядок"; следовательно, единственная перестановка, не содержащая инверсий, — это рассортированная перестановка $1\,2\ldots n$. Такая связь с сортировкой и является главной причиной нашего интереса к инверсиям, хотя это понятие уже использовалось, когда мы анализировали алгоритм динамического распределения памяти (см. упр. 2.2.2–9).

Понятие инверсии ввел Г. Крамер в 1750 году в связи со своим замечательным правилом решения линейных уравнений [*Intr. à l'Analyse des Lignes Courbes Algébriques* (Geneva, 1750), 657–659; см. также Thomas Muir, *Theory of Determinants* **1** (1906), 11–14]. В сущности, он определил детерминант $n\times n$-матрицы следующим образом:

$$\det\begin{pmatrix} x_{11} & x_{12} & \ldots & x_{1n}\\ \vdots & \vdots & & \vdots\\ x_{n1} & x_{n2} & \ldots & x_{nn}\end{pmatrix}=\sum(-1)^{\operatorname{inv}(a_1a_2\ldots a_n)}x_{1a_1}x_{2a_2}\ldots x_{na_n},$$

где сумма берется по всем перестановкам $a_1\,a_2\ldots a_n$ из $\{1,2,\ldots,n\}$, а $\operatorname{inv}(a_1\,a_2\ldots a_n)$ — число инверсий в перестановке.

Таблица инверсий $b_1\,b_2\ldots b_n$ перестановки $a_1\,a_2\ldots a_n$ образуется, если определить b_j как число элементов слева от j, которые больше, чем j. Другими словами, b_j — число инверсий, второй элемент которых равен j. Отсюда, например, следует, что таблицей инверсий перестановки

$$5\,9\,1\,8\,2\,6\,4\,7\,3 \tag{1}$$

будет

$$2\,3\,6\,4\,0\,2\,2\,1\,0, \tag{2}$$

поскольку 5 и 9 расположены левее 1; 5, 9, 8 — левее 2 и т. д. Всего эта перестановка имеет 20 инверсий. По определению числа b_j всегда удовлетворяют неравенствам

$$0 \le b_1 \le n-1, \quad 0 \le b_2 \le n-2, \quad \ldots, \quad 0 \le b_{n-1} \le 1, \quad b_n = 0. \tag{3}$$

Пожалуй, наиболее важный факт, касающийся перестановок и установленный Маршаллом Холлом (Marshall Hall), состоит в том, что *таблица инверсий единственным образом определяет соответствующую перестановку.* [См. *Proc. Symp. Applied Math.* **6** (American Math. Society, 1956), 203.] Из любой таблицы инверсии $b_1\, b_2 \ldots b_n$, удовлетворяющей условиям (3), можно однозначно восстановить перестановку, которая ее породила, путем последовательного определения относительного расположения элементов $n, n-1, \ldots, 1$ (в этом порядке). Например, перестановку, соответствующую (2), можно построить следующим образом: выпишем число 9, затем разместим 8 после 9, поскольку $b_8 = 1$. Аналогично установим 7 после и 8, и 9, поскольку $b_7 = 2$. Так как $b_6 = 2$, то 6 должно следовать за двумя уже выписанными нами числами; таким образом, имеем промежуточный результат

$$9\,8\,6\,7.$$

Продолжим, приписав 5 левее, поскольку $b_5 = 0$; поместим 4 вслед за четырьмя из уже записанных чисел, а 3 — после шести выписанных чисел (т. е. в правый конец) и получим

$$5\,9\,8\,6\,4\,7\,3.$$

Вставив аналогичным образом 2 и 1, придем к (1).

Такое соответствие важно, потому что часто можно заменить задачу, сформулированную в терминах перестановок, эквивалентной ей задачей, сформулированной в терминах таблиц инверсий, которая, возможно, решается проще. Рассмотрим, например, самый простой вопрос: сколько существует перестановок множества $\{1, 2, \ldots, n\}$? Ответ должен быть равен числу всевозможных таблиц инверсий, а их легко пересчитать, так как b_1 можно выбрать n способами, b_2 независимо от b_1 — $n-1$ способами, ..., а b_n — единственным способом; итого получаем $n(n-1)\ldots 1 = n!$ различных таблиц инверсий. Таблицы инверсий пересчитать легче, потому что значения b_k полностью не зависят друг от друга, в то время как a_k должны быть все различны.

В разделе 1.2.10 мы рассматривали задачу о числе локальных максимумов перестановки, если читать ее справа налево. Иными словами, требовалось подсчитать количество элементов, каждый из которых больше любого из следующих после него. (Например, правосторонние максимумы в (1) — это 3, 7, 8 и 9.) Оно равно количеству индексов j, таких, что b_j имеет максимальное значение $n-j$. Так как b_1 будет равно $n-1$ с вероятностью $1/n$ и (независимо) b_2 будет равно $n-2$ с вероятностью $1/(n-1)$ и т. д., из рассмотрения инверсий ясно, что среднее число правосторонних максимумов равно

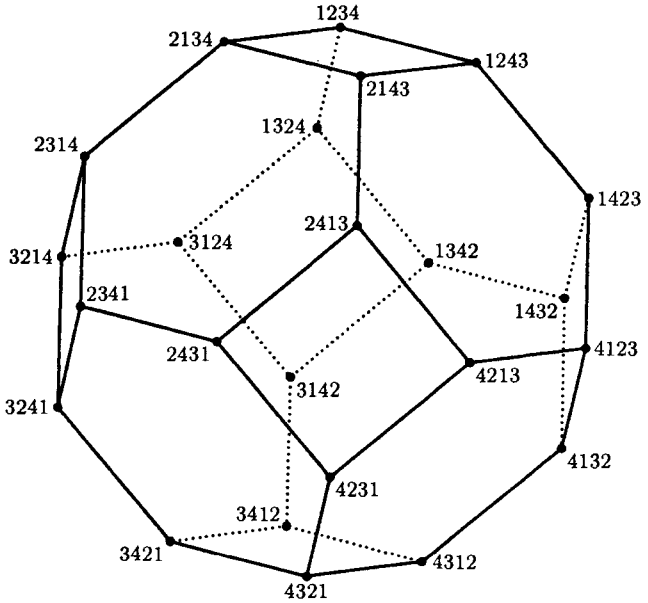


Рис. 1. Усеченный октаэдр, на котором показано изменение числа инверсий, когда меняются местами два соседних элемента перестановки.

$$\frac{1}{n} + \frac{1}{n-1} + \cdots + \frac{1}{1} = H_n.$$

Аналогичным способом легко получить и соответствующую производящую функцию.

Ясно, что если поменять местами *соседние* элементы перестановки, то общее число инверсий увеличится или уменьшится на единицу. На рис. 1 показаны 24 перестановки множества $\{1, 2, 3, 4\}$; линиями соединены перестановки, отличающиеся одна от другой положением двух соседних элементов; двигаясь вдоль линии вниз, мы увеличиваем число инверсий на единицу. Следовательно, число инверсий в перестановке π равно длине нисходящего пути из 1234 в π на рис. 1; все такие пути должны иметь одинаковую длину.

Заметим, что эту диаграмму можно рассматривать как трехмерное твердое тело — "усеченный октаэдр", имеющий 8 шестиугольных и 6 квадратных граней. Это один из классических правильных многогранников, которые исследовал еще Архимед (см. упр. 10).

Не следует путать "инверсии" перестановок (inversions of a permutation) с *обратными* перестановками (*inverse* of a permutation). Вспомним, что перестановку можно записывать в виде двух строк:

$$\begin{pmatrix} 1 & 2 & 3 & \dots & n \\ a_1 & a_2 & a_3 & \dots & a_n \end{pmatrix}. \tag{4}$$

Обратной называется перестановка $a'_1\, a'_2\, a'_3 \dots a'_n$, которая получается, если в (4) поменять местами строки, а затем упорядочить столбцы в порядке возрастания по верхним элементам:

$$\begin{pmatrix} a_1 & a_2 & a_3 & \dots & a_n \\ 1 & 2 & 3 & \dots & n \end{pmatrix} = \begin{pmatrix} 1 & 2 & 3 & \dots & n \\ a'_1 & a'_2 & a'_3 & \dots & a'_n \end{pmatrix}. \tag{5}$$

Например, обратной к перестановке 5 9 1 8 2 6 4 7 3 будет перестановка 3 5 9 7 1 6 8 4 2, так как

$$\begin{pmatrix} 5 & 9 & 1 & 8 & 2 & 6 & 4 & 7 & 3 \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \end{pmatrix} = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 3 & 5 & 9 & 7 & 1 & 6 & 8 & 4 & 2 \end{pmatrix}.$$

Можно дать другое определение обратной перестановки: $a'_j = k$ тогда и только тогда, когда $a_k = j$.

Понятие обратной перестановки впервые ввел Х. А. Роте (H. A. Rothe) [в *Sammlung combinatorisch-analytischer Abhandlungen*, edited by K. F. Hindenburg, **2** (Leipzig, 1800), 263–305]. Он заметил интересную связь между обратными перестановками и инверсиями: *обратная перестановка содержит ровно столько инверсий, сколько исходная.* Роте дал не самое простое доказательство этого факта, но оно поучительно и притом довольно красиво. Построим таблицу размером $n \times n$, наподобие шахматной доски, в которой точки стоят в j-й клетке i-й строки, если $a_i = j$. После этого расставим крестики во всех клетках, снизу от которых (в том же столбце) и справа (в той же строке) есть точки. Например, для 5 9 1 8 2 6 4 7 3 диаграмма будет выглядеть следующим образом:

×	×	×	×	•				
×	×	×	×		×	×	×	•
•								
	×	×	×		×	×	•	
	•							
		×	×		•			
		×	•					
		×				•		
		•						

.

Количество крестиков равно числу инверсий, так как нетрудно видеть, что b_j равно числу крестиков в j-м столбце. Если мы теперь транспонируем диаграмму (поменяв ролями строки и столбцы), то получим диаграмму для обратной по отношению к исходной перестановки; значит, число крестиков (число инверсий) одинаково в обоих случаях. Роте использовал данный факт для доказательства того, что детерминант матрицы не меняется при транспонировании.

Для анализа некоторых алгоритмов сортировки необходимо знать число перестановок n элементов, содержащих ровно k инверсий. Обозначим это число через $I_n(k)$; в табл. 1 приведено несколько первых значений данной функции.

Из таблицы инверсий $b_1 b_2 \dots b_n$ ясно, что $I_n(0) = 1$, $I_n(1) = n - 1$ и что выполняется свойство симметрии:

$$I_n\left(\binom{n}{2} - k\right) = I_n(k). \tag{6}$$

Далее, так как значения b_k можно выбирать независимо одно от другого, нетрудно видеть, что производящая функция

$$G_n(z) = I_n(0) + I_n(1)z + I_n(2)z^2 + \cdots \tag{7}$$

Таблица 1

ПЕРЕСТАНОВКА С k ИНВЕРСИЯМИ

n	$I_n(0)$	$I_n(1)$	$I_n(2)$	$I_n(3)$	$I_n(4)$	$I_n(5)$	$I_n(6)$	$I_n(7)$	$I_n(8)$	$I_n(9)$	$I_n(10)$	$I_n(11)$
1	1	0	0	0	0	0	0	0	0	0	0	0
2	1	1	0	0	0	0	0	0	0	0	0	0
3	1	2	2	1	0	0	0	0	0	0	0	0
4	1	3	5	6	5	3	1	0	0	0	0	0
5	1	4	9	15	20	22	20	15	9	4	1	0
6	1	5	14	29	49	71	90	101	101	90	71	49

удовлетворяет соотношению $G_n(z) = (1 + z + \cdots + z^{n-1})G_{n-1}(z)$; следовательно, она имеет довольно простой вид, как показано О. Родригесом (O. Rodrigues) [*J. de Math.* **4** (1839), 236–240]:

$$(1 + z + \cdots + z^{n-1})\ldots(1 + z)(1) = (1 - z^n)\ldots(1 - z^2)(1 - z)/(1 - z)^n. \tag{8}$$

С помощью этой производящей функции можно легко продолжить табл. 1 и убедиться, что числа, расположенные под ступенчатой линией в таблице, удовлетворяют соотношению

$$I_n(k) = I_n(k - 1) + I_{n-1}(k), \quad \text{где} \quad k < n. \tag{9}$$

(Для чисел, находящихся *над* ступенчатой линией, это соотношение *не* выполняется.) Более сложные рассуждения (см. упр. 14) показывают, что на самом деле справедливы формулы

$$\begin{aligned}
I_n(2) &= \binom{n}{2} - 1, && n \ge 2;\\
I_n(3) &= \binom{n+1}{3} - \binom{n}{1}, && n \ge 3;\\
I_n(4) &= \binom{n+2}{4} - \binom{n+1}{2}, && n \ge 4;\\
I_n(5) &= \binom{n+3}{5} - \binom{n+2}{3} + 1, && n \ge 5.
\end{aligned}$$

Общая формула для $I_n(k)$ содержит около $1.6\sqrt{k}$ слагаемых:

$$\begin{aligned}
I_n(k) = {} & \binom{n+k-2}{k} - \binom{n+k-3}{k-2} + \binom{n+k-6}{k-5} + \binom{n+k-8}{k-7} - \cdots\\
& + (-1)^j\left(\binom{n+k-u_j-1}{k-u_j} + \binom{n+k-u_j-j-1}{k-u_j-j}\right) + \cdots, \quad n \ge k,
\end{aligned} \tag{10}$$

где $u_j = (3j^2 - j)/2$ — так называемое “пентагональное число”.

Разделив $G_n(z)$ на $n!$, получим производящую функцию $g_n(z)$ распределения вероятностей числа инверсий в случайной перестановке n элементов. Она равна произведению

$$g_n(z) = h_1(z)h_2(z)\ldots h_n(z), \tag{11}$$

где $h_k(z) = (1 + z + \cdots + z^{k-1})/k$ — производящая функция равномерного распределения случайной величины, принимающей целые неотрицательные значения, меньшие k. Отсюда*

$$\begin{aligned}
\operatorname{mean}(g_n) &= \operatorname{mean}(h_1) + \operatorname{mean}(h_2) + \cdots + \operatorname{mean}(h_n) \\
&= 0 + \frac{1}{2} + \cdots + \frac{n-1}{2} = \frac{n(n-1)}{4}; \qquad (12)\\
\operatorname{var}(g_n) &= \operatorname{var}(h_1) + \operatorname{var}(h_2) + \cdots + \operatorname{var}(h_n) \\
&= 0 + \frac{1}{4} + \cdots + \frac{n^2-1}{12} = \frac{n(2n+5)(n-1)}{72}. \qquad (13)
\end{aligned}$$

Таким образом, среднее число инверсий довольно велико — около $\frac{1}{4}n^2$; стандартное отклонение также весьма велико — около $\frac{1}{6}n^{3/2}$.

В качестве интересного завершения данной темы рассмотрим одно замечательное открытие, принадлежащее П. А. Мак-Магону (P. A. MacMahon) [*Amer. J. Math.* **35** (1913), 281–322]. Определим *индекс* перестановки $a_1\, a_2 \ldots a_n$ как сумму всех j, таких, что $a_j > a_{j+1}$, $1 \le j < n$. Например, индекс перестановки 5 9 1 8 2 6 4 7 3 равен $2+4+6+8 = 20$. Индекс случайно совпал с числом инверсий. Если составить список всех приведенных ниже 24 перестановок множества $\{1, 2, 3, 4\}$, то получится, что *число перестановок, имеющих данный индекс k, равно числу перестановок, имеющих k инверсий.*

Перестановка	Индекс	Количество инверсий	Перестановка	Индекс	Количество инверсий
1 2 3 4	0	0	3|1 2 4	1	2
1 2 4|3	3	1	3|1 4|2	4	3
1 3|2 4	2	1	3|2|1 4	3	3
1 3 4|2	3	2	3|2 4|1	4	4
1 4|2 3	2	2	3 4|1 2	2	4
1 4|3|2	5	3	3 4|2|1	5	5
2|1 3 4	1	1	4|1 2 3	1	3
2|1 4|3	4	2	4|1 3|2	4	4
2 3|1 4	2	2	4|2|1 3	3	4
2 3 4|1	3	3	4|2 3|1	4	5
2 4|1 3	2	3	4|3|1 2	3	5
2 4|3|1	5	4	4|3|2|1	6	6

На первый взгляд, этот факт может показаться почти очевидным, однако после некоторых размышлений он начинает казаться чуть ли не мистическим, и не видно никакого простого прямого его доказательства. Мак-Магон нашел следующее остроумное косвенное доказательство. Пусть $\operatorname{ind}(a_1\, a_2 \ldots a_n)$ — индекс перестановки $a_1\, a_2 \ldots a_n$ и соответствующая производящая функция есть

$$H_n(z) = \sum z^{\operatorname{ind}(a_1\, a_2 \ldots a_n)}, \qquad (14)$$

* Здесь и далее $\operatorname{mean}(g)$ означает среднее значение случайной величины g, а $\operatorname{var}(g)$ — дисперсию величины g. — *Прим. перев.*

где сумма в (14) берется по всем перестановкам множества $\{1, 2, \ldots, n\}$. Требуется доказать, что $H_n(z) = G_n(z)$. Для этого определим взаимно однозначное соответствие между n-мерными строками $(q_1, q_2, \ldots, q_n)$ неотрицательных целых чисел, с одной стороны, и упорядоченными парами n-мерных строк

$$\big((a_1, a_2, \ldots, a_n),\ (p_1, p_2, \ldots, p_n)\big),$$

с другой стороны, где $a_1\, a_2 \ldots a_n$ — суть перестановка множества индексов $\{1, 2, \ldots, n\}$ и $p_1 \geq p_2 \geq \cdots \geq p_n \geq 0$. Это соответствие будет удовлетворять условию

$$q_1 + q_2 + \cdots + q_n = \operatorname{ind}(a_1\, a_2 \ldots a_n) + (p_1 + p_2 + \cdots + p_n). \tag{15}$$

Производящая функция $\sum z^{q_1+q_2+\cdots+q_n}$, где сумма берется по всем n-мерным строкам неотрицательных целых чисел $(q_1, q_2, \ldots, q_n)$, равна $Q_n(z) = 1/(1-z)^n$; а производящая функция $\sum z^{p_1+p_2+\cdots+p_n}$, где сумма берется по всем n-мерным строкам целых чисел $(p_1, p_2, \ldots, p_n)$, таких, что $p_1 \geq p_2 \geq \cdots \geq p_n \geq 0$, равна

$$P_n(z) = 1/(1-z)(1-z^2)\ldots(1-z^n), \tag{16}$$

как показано в упр. 15. Существование взаимно однозначного соответствия, которое удовлетворяет условию (15) и которое мы собираемся установить, доказывает равенство $Q_n(z) = H_n(z)P_n(z)$, т. е.

$$H_n(z) = Q_n(z)/P_n(z). \tag{17}$$

Но $Q_n(z)/P_n(z)$ и есть $G_n(z)$, как следует из (8).

Требуемое соответствие определяется с помощью простой процедуры сортировки. Любая n-мерная строка $(q_1, q_2, \ldots, q_n)$ может быть устойчиво реорганизована в порядке невозрастания $q_{a_1} \geq q_{a_2} \geq \cdots \geq q_{a_n}$, где $a_1\, a_2 \ldots a_n$ — перестановка, такая, что $q_{a_j} = q_{a_{j+1}}$ и подразумевает $a_j < a_{j+1}$. Установим $(p_1, p_2, \ldots, p_n) = (q_{a_1}, q_{a_2}, \ldots, q_{a_n})$ и затем для $1 \leq j < n$ вычтем 1 из каждого $p_1, \ldots, p_j$ для таких j, которые соответствуют $a_j > a_{j+1}$. По-прежнему $p_1 \geq p_2 \geq \cdots \geq p_n$, поскольку p_j обязательно превышает p_{j+1}, если только $a_j > a_{j+1}$. Полученная в результате пара $\big((a_1, a_2, \ldots, a_n), (p_1, p_2, \ldots, p_n)\big)$ удовлетворяет условию (15), поскольку суммарное уменьшение элементов p равно $\operatorname{ind}(a_1\, a_2 \ldots a_n)$. Например, если $n = 9$ и $(q_1, \ldots, q_9) = (3, 1, 4, 1, 5, 9, 2, 6, 5)$, то получим $a_1 \ldots a_9 = 6\,8\,5\,9\,3\,1\,7\,2\,4$ и $(p_1, \ldots, p_9) = (5, 2, 2, 2, 2, 2, 1, 1, 1)$.

Несложно вернуться к $(q_1, q_2, \ldots, q_n)$, когда заданы $a_1\, a_2 \ldots a_n$ и $(p_1, p_2, \ldots, p_n)$ (см. упр. 17). Итак, желаемое соответствие установлено, теорема Мак-Магона об индексах доказана.

Д. Фоата (D. Foata) и М. П. Шуценберже (M. P. Schützenberger) отыскали неожиданное расширение теоремы Мак-Магона через 65 лет после его первой публикации. *Число перестановок n элементов, которые имеют k инверсий и индекс l, равно числу перестановок, которые имеют l инверсий и индекс k.* Фактически Фоата и Шуценберже нашли простое однозначное соответствие между перестановками первого и второго типов (см. упр. 25).

УПРАЖНЕНИЯ

1. [*10*] Какова таблица инверсий для перестановки 2 7 1 8 4 5 9 3 6? Какой перестановке соответствует таблица инверсий 5 0 1 2 1 2 0 0?

2. [*M20*] Классическая задача Иосифа формулируется следующим образом (см. также упр. 1.3.2–22): n рабов вначале стоят по кругу; после того как m-го раба казнят, круг смыкается, затем опять казнят m-го раба и так до тех пор, пока всех рабов не постигнет эта печальная участь. Таким образом, порядок, в котором рабы подвергаются наказанию, представляет собой перестановку множества $\{1, 2, \ldots, n\}$. Например, если $n = 8$ и $m = 4$, порядок будет иметь вид 5 4 6 1 3 8 7 2 (первым казнили 5-го раба и т. д.); таблица инверсий для этой перестановки имеет вид 3 6 3 1 0 0 1 0.

Найдите простое рекуррентное соотношение для элементов $b_1\, b_2 \ldots b_n$ таблицы инверсий в общей задаче Иосифа для n рабов, если каждого m-го раба казнят.

3. [*18*] Пусть перестановке $a_1\, a_2 \ldots a_n$ соответствует таблица инверсий $b_1\, b_2 \ldots b_n$. Какой перестановке $\overline{a}_1\, \overline{a}_2 \ldots \overline{a}_n$ соответствует таблица инверсий

$$(n-1-b_1)(n-2-b_2)\ldots(0-b_n)\,?$$

▶ **4.** [*20*] Придумайте пригодный для компьютерной реализации алгоритм, который по данной таблице инверсий $b_1\, b_2 \ldots b_n$, удовлетворяющей условиям (3), строил бы соответствующую перестановку $a_1\, a_2 \ldots a_n$. [*Указание.* Вспомните методы работы со связями в памяти.]

5. [*35*] Для выполнения алгоритма из упр. 4 на типичном компьютере потребуется время, приблизительно пропорциональное $n + b_1 + \cdots + b_n$, а это в среднем равно $\Theta(n^2)$. Можно ли создать алгоритм, порядок времени выполнения которого был бы существенно меньше, чем n^2?

▶ **6.** [*26*] Придумайте алгоритм вычисления таблицы инверсий $b_1\, b_2 \ldots b_n$, соответствующей данной перестановке $a_1\, a_2 \ldots a_n$ множества $\{1, 2, \ldots, n\}$, время работы которого на типичном компьютере было бы порядка $n \log n$.

7. [*20*] Помимо таблицы $b_1\, b_2 \ldots b_n$, определенной в этом упражнении, можно определить некоторые типы таблиц инверсий, соответствующих данной перестановке $a_1\, a_2 \ldots a_n$ множества $\{1, 2, \ldots, n\}$. В этом упражнении мы рассмотрим три других типа таблиц инверсий, которые возникают в приложениях.

Пусть c_j — число инверсий, *первая* компонента которых равна j, т. е. число элементов, меньших j и расположенных *правее* j. [Перестановке (1) соответствует таблица 0 0 0 1 4 2 1 5 7; ясно, что $0 \le c_j < j$.] Пусть $B_j = b_{a_j}$ и $C_j = c_{a_j}$.

Покажите, что при $1 \le j \le n$ справедливы неравенства $0 \le B_j < j$ и $0 \le C_j \le n - j$; покажите также, что перестановку $a_1\, a_2 \ldots a_n$ можно однозначно определить, если задана таблица $c_1\, c_2 \ldots c_n$ или $B_1\, B_2 \ldots B_n$, или $C_1\, C_2 \ldots C_n$.

8. [*M24*] Сохраним обозначения, принятые в упр. 7. Пусть $a'_1\, a'_2 \ldots a'_n$ — перестановка, обратная к $a_1\, a_2 \ldots a_n$, и пусть соответствующие ей таблицы инверсий — $b'_1\, b'_2 \ldots b'_n$, $c'_1\, c'_2 \ldots c'_n$, $B'_1\, B'_2 \ldots B'_n$ и $C'_1\, C'_2 \ldots C'_n$. Найдите как можно больше интересных соотношений между числами a_j, b_j, c_j, B_j, C_j, a'_j, b'_j, c'_j, B'_j, C'_j.

▶ **9.** [*M21*] Докажите, что в обозначениях, принятых в упр. 7, перестановка $a_1\, a_2 \ldots a_n$ представляет собой инволюцию (т. е. обратна самой себе) тогда и только тогда, когда $b_j = C_j$ при $1 \le j \le n$.

10. [*HM20*] Рассмотрите представленный на рис. 1 октаэдр как многогранник в трехмерном пространстве. Чему равен диаметр усеченного октаэдра (расстояние между вершинами 1234 и 4321), если все ребра имеют единичную длину?

11. [*M25*] Если $\pi = a_1\, a_2 \ldots a_n$ представляет собой перестановку множества $\{1, 2, \ldots, n\}$, то обозначим как

$$E(\pi) = \{(a_i, a_j) \mid i < j,\ a_i > a_j\}$$

множество ее инверсий, а как

$$\bar{E}(\pi) = \{(a_i, a_j) \mid i > j,\ a_i > a_j\}$$

множество ее "неинверсий".

a) Докажите, что $E(\pi)$ и $\bar{E}(\pi)$ транзитивны. (Множество S упорядоченных пар называется *транзитивным*, если для любых (a, b) и (b, c), принадлежащих S, пара (a, c) также принадлежит S.)

b) Обратно, пусть E — любое транзитивное подмножество множества $T = \{(x, y) \mid 1 \le y < x \le n\}$, дополнение которого $\bar{E} = T \setminus E$ также транзитивно. Докажите, что существует перестановка π, такая, что $E(\pi) = E$.

12. [*M28*] Используя обозначения, принятые в предыдущем упражнении, докажите, что если π_1 и π_2 — перестановки, а E — минимальное транзитивное множество, содержащее $E(\pi_1) \cup E(\pi_2)$, то $\bar{E}$ — также транзитивное множество. [Следовательно, если мы говорим, что π_1 находится "над" π_2, когда $E(\pi_1) \subseteq E(\pi_2)$, то определена *решетка* перестановок; существует единственная "самая низкая" перестановка, находящаяся "над" двумя данными перестановками. Диаграмма решетки при $n = 4$ представлена на рис. 1.]

13. [*M23*] Известно, что в разложении определителя половина членов выписывается со знаком "+", а половина — со знаком "−". Другими словами, при $n \ge 2$ перестановок с *четным* числом инверсий ровно столько же, сколько с *нечетным*. Покажите, что вообще при $n \ge m$ количество перестановок с числом инверсий, конгруэнтным t по модулю m, равно $n!/m$, независимо от того, каково целое число t.

14. [*M24*] (Ф. Франклин (F. Franklin).) Разбиение числа n на k различных частей — это представление n в виде суммы $n = p_1 + p_2 + \cdots + p_k$, где $p_1 > p_2 > \cdots > p_k > 0$. Например, разбиения числа 7 на различные части таковы: 7, 6 + 1, 5 + 2, 4 + 3, 4 + 2 + 1. Пусть $f_k(n)$ — число разбиений n на k различных частей. Докажите, что $\sum_k (-1)^k f_k(n) = 0$, если только n не представляется в виде $(3j^2 \pm j)/2$ при некотором неотрицательном целом j; в этом случае сумма равна $(-1)^j$. Например, для $n = 7$ сумма равна $-1 + 3 - 1 = 1$, потому что $7 = (3 \cdot 2^2 + 2)/2$. [*Указание.* Представьте разбиения в виде массива точек, в i-й строке которого имеется p_i точек, $1 \le i \le k$. Найдите наименьшее j, такое, что $p_{j+1} < p_j - 1$, и обведите крайние справа точки первых j строк. Если $j < p_k$, то эти j точек можно, как правило, изъять из массива, повернуть на 45° и поместить в новую, $(k+1)$-ю, строку. С другой стороны, если $j \ge p_k$, то обычно можно изъять из массива k-ю строку точек, повернуть ее на 45° и поместить справа от обведенных точек (рис. 2). В результате в большинстве случаев разбиения с четным числом строк и разбиения с нечетным числом строк группируются в пары; таким образом, в сумме следует учитывать только непарные разбиения.]

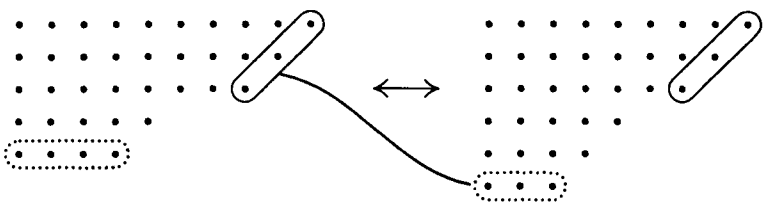


Рис. 2. Соответствие Франклина между разбиениями на различные части.

Замечание. Как следствие получаем формулу Эйлера:

$$(1 - z)(1 - z^2)(1 - z^3)\ldots = 1 - z - z^2 + z^5 + z^7 - z^{12} - z^{15} + \cdots$$
$$= \sum_{-\infty < j < \infty} (-1)^j z^{(3j^2 + j)/2}.$$

Поскольку производящая функция для обычных разбиений (не обязательно на различные части) равна $\sum p(n)z^n = 1/(1-z)(1-z^2)(1-z^3)\ldots$, получаем неочевидное рекуррентное соотношение для числа разбиений:

$$p(n) = p(n-1) + p(n-2) - p(n-5) - p(n-7) + p(n-12) + p(n-15) - \cdots.$$

15. [*M23*] Докажите, что соотношение (16) — это производящая функция для числа разбиений не более чем на n частей, т. е. докажите, что коэффициент при z^m в $1/(1-z)(1-z^2)\ldots(1-z^n)$ равен числу способов представления m в виде суммы $m = p_1 + p_2 + \cdots + p_n$, где $p_1 \geq p_2 \geq \cdots \geq p_n \geq 0$. [*Указание.* Нарисуйте точки, как в упр. 14, и покажите, что существует взаимно однозначное соответствие между n-мерными строками чисел $(p_1, p_2, \ldots, p_n)$, такими, что $p_1 \geq p_2 \geq \cdots \geq p_n \geq 0$, и последовательностями $(P_1, P_2, P_3, \ldots)$, такими, что $n \geq P_1 \geq P_2 \geq P_3 \geq \cdots \geq 0$, обладающее тем свойством, что $p_1 + p_2 + \cdots + p_n = P_1 + P_2 + P_3 + \cdots$. Иными словами, покажите, что разбиениям не более чем на n частей соответствуют разбиения на части, не превосходящие n.]

16. [*M25*] (Л. Эйлер.) Докажите следующие тождества, интерпретируя обе части соотношений в терминах разбиений:

$$\prod_{k\geq 0} \frac{1}{(1-q^k z)} = \frac{1}{(1-z)(1-qz)(1-q^2z)\ldots}$$
$$= 1 + \frac{z}{1-q} + \frac{z^2}{(1-q)(1-q^2)} + \cdots = \sum_{n\geq 0} z^n \Big/ \prod_{1\leq k\leq n} (1-q^k);$$

$$\prod_{k\geq 0} (1+q^k z) = (1+z)(1+qz)(1+q^2z)\ldots$$
$$= 1 + \frac{z}{1-q} + \frac{z^2 q}{(1-q)(1-q^2)} + \cdots = \sum_{n\geq 0} z^n q^{n(n-1)/2} \Big/ \prod_{1\leq k\leq n} (1-q^k).$$

17. [*20*] Каковы 24 тетрады (q_1, q_2, q_3, q_4), для которых в соответствии Мак-Магона, определенном в конце этого раздела, $(p_1, p_2, p_3, p_4) = (0, 0, 0, 0)$?

18. [*M30*] (T. Hibbard, *CACM* **6** (1963), 210.)

Пусть $n > 0$ и последовательность 2^n n-разрядных целых чисел $X_0, \ldots, X_{2^n-1}$ получена случайным образом, причем каждый разряд каждого числа независимо от других разрядов принимает значение 1 с вероятностью p. Рассмотрим последовательность $X_0 \oplus 0$, $X_1 \oplus 1$, $\ldots$, $X_{2^n-1} \oplus (2^n - 1)$, где $\oplus$ — операция "исключающее или" над двоичными представлениями. Так, если $p = 0$, то последовательность будет такой: $0, 1, \ldots, 2^n - 1$; если $p = 1$, то она будет такой: $2^n - 1, \ldots, 1, 0$; если же $p = \frac{1}{2}$, то каждый элемент последовательности — случайное число между 0 и $2^n - 1$. Вообще же, при разных p это хороший способ получения последовательности случайных целых чисел со смещенным числом инверсий, в то время как распределение элементов последовательности, рассматриваемой как единое целое, равномерно (uniform) в том смысле, что все n-разрядные целые двоичные числа будут иметь одинаковые распределения. Определите среднее число инверсий в такой последовательности как функцию от вероятности p.

19. [*M28*] (К. Мейер (C. Meyer).) Если m и n — взаимно простые числа, то известно, что последовательность $(m \bmod n)(2m \bmod n)\ldots((n-1)m \bmod n)$ представляет собой перестановку множества $\{1, 2, \ldots, n-1\}$. Покажите, что число инверсий этой перестановки может быть выражено в терминах сумм Дедекинда (см. п. 3.3.3).

20. [*M43*] Следующее знаменитое тождество, принадлежащее Якоби [*Fundamenta Nova Theoriæ Functionum Ellipticarum* (1829), §64], лежит в основе многих замечательных соотношений, содержащих эллиптические функции:

$$\prod_{k\ge1}(1-u^kv^{k-1})(1-u^{k-1}v^k)(1-u^kv^k)$$
$$= (1-u)(1-v)(1-uv)(1-u^2v)(1-uv^2)(1-u^2v^2)\ldots$$
$$= 1-(u+v)+(u^3v+uv^3)-(u^6v^3+u^3v^6)+\cdots$$
$$= \sum_{-\infty<j<+\infty}(-1)^j u^{\binom{j}{2}} v^{\binom{j+1}{2}}.$$

Если, например, положить $u=z$, $v=z^2$, то получится формула Эйлера из упр. 14. Если положить $z=\sqrt{u/v}$, $q=\sqrt{uv}$, то получим

$$\prod_{k\ge1}(1-q^{2k-1}z)(1-q^{2k-1}z^{-1})(1-q^{2k}) = \sum_{-\infty<n<\infty}(-1)^n z^n q^{n^2}.$$

Существует ли комбинаторное доказательство тождества Якоби, аналогичное доказательству Франклина для частного случая из упр. 14? (Таким образом, нужно рассмотреть "комплексные разбиения"

$$m+ni=(p_1+q_1i)+(p_2+q_2i)+\cdots+(p_k+q_ki),$$

где p_j+q_ji — различные ненулевые комплексные числа; p_j, q_j, — неотрицательные целые числа, причем $|p_j-q_j|\le 1$. Согласно тождеству Якоби число таких представлений с четными k равно числу представлений с нечетными k, если только m и n не являются соседними треугольными числами!) Какими еще замечательными свойствами обладают комплексные разбиения?

▸ **21.** [*M25*] (Г. Д. Кнотт (G. D. Knott).) Покажите, что перестановку $a_1\ldots a_n$ можно получить с помощью стека (см. упр. 2.2.1–5 или 2.3.1–6) тогда и только тогда, когда $C_j\le C_{j+1}+1$ при $1\le j<n$ (см. обозначения в упр. 7).

22. [*M26*] Задана перестановка $a_1\,a_2\ldots a_n$ множества $\{1,2,\ldots,n\}$. Пусть h_j — число индексов $i<j$, таких, что $a_i\in\{a_j+1,a_j+2,\ldots,a_{j+1}\}$. (Если $a_{j+1}<a_j$, элементы этого множества "оборачиваются" от n к 1. Когда $j=n$, используется множество $\{a_n+1,a_n+2,\ldots,n\}$.) Например, перестановка 5 9 1 8 2 6 4 7 3 приводит к $h_1\ldots h_9=0\,0\,1\,2\,1\,4\,2\,4\,6$.

a) Докажите, что $a_1\,a_2\ldots a_n$ можно восстановить по числам $h_1\,h_2\ldots h_n$.

b) Докажите, что $h_1+h_2+\cdots+h_n$ — суть индексы $a_1\,a_2\ldots a_n$.

▸ **23.** [*M27*] (*Русская рулетка.*) Группа из n человек, приговоренных к смерти, которые отдают предпочтение теории вероятности перед теорией чисел, могла бы, рассевшись по кругу и несколько модифицировав метод Иосифа (упр. 2), попробовать такой способ самоубийства. Первый приговоренный нажимает спусковой крючок револьвера, направив его себе в висок; с вероятностью p произойдет роковой выстрел, и он покинет круг. Затем револьвер переходит ко второму приговоренному и он делает то же самое. Далее сюжет повторяется по кругу с постоянной вероятностью $p>0$ до тех пор, пока будет кому его продолжать.

Пусть $a_j=k$, если k-му приговоренному выпал j-й роковой выстрел. Докажите, что порядок "выбывания" $a_1\,a_2\ldots a_n$ появится с вероятностью, которая является функцией только n, p и индекса дуальной перестановки $(n+1-a_n)\ldots(n+1-a_2)\,(n+1-a_1)$. Какой порядок "выбывания" наименее вероятен?

24. [*M26*] Для заданного множество целых чисел $t(1)\,t(2)\ldots t(n)$, где $t(j)\ge j$, *обобщенный индекс* перестановки $a_1\,a_2\ldots a_n$ равен сумме всех нижних индексов j, таких, что $a_j>t(a_{j+1})$, плюс общее число инверсий, таких, что $i<j$ и $t(a_j)\ge a_i>a_j$. Значит, если $t(j)=j$ для всех j, обобщенный индекс совпадает с обычным индексом, но при $t(j)\ge n$ для всех j это будет число инверсий. Докажите, что число инверсий, для которых обобщенный индекс равен k, — то же самое, что и число перестановок, которые

имеют k инверсий. [*Указание.* Покажите, что, если взять любую перестановку $a_1 \ldots a_{n-1}$ множества $\{1, \ldots, n-1\}$ и поместить число n на все возможные места, обобщенный индекс увеличится на $\{0, 1, \ldots, n-1\}$ в некотором порядке.]

▶ **25.** [*M30*] (Фоата и Шуценберже.) Пусть существует перестановка $\alpha = a_1 \ldots a_n$; обозначим через $\mathrm{ind}(\alpha)$ ее индекс, а через $\mathrm{inv}(\alpha)$ — число ее инверсий.

a) Определите взаимно однозначное соответствие, которое преобразует перестановку α множества $\{1, \ldots, n\}$ в перестановку $f(\alpha)$, имеющую такие два свойства: (i) $\mathrm{ind}(f(\alpha)) = \mathrm{inv}(\alpha)$; (ii) для $1 \le j < n$ число j появляется слева от $j+1$ в $f(\alpha)$ тогда и только тогда, когда оно появляется слева от $j+1$ в α. Какая перестановка при этом соответствует $f(\alpha)$, если $\alpha = 1\,9\,8\,2\,6\,3\,7\,4\,5$? Для какой перестановки $f(\alpha) = 1\,9\,8\,2\,6\,3\,7\,4\,5$? [*Указание.* Если $n > 1$, напишите $\alpha = x_1\alpha_1x_2\alpha_2 \ldots x_k\alpha_k a_n$, где $x_1, \ldots, x_k$ — все элементы, меньшие, чем a_n, если $a_1 < a_n$; в противном случае $x_1, \ldots, x_k$ — все элементы, большие, чем a_n; другие элементы появятся в цепочках (возможно, пустых) $\alpha_1, \ldots, \alpha_k$. Сравните число инверсий $h(\alpha) = \alpha_1x_1\alpha_2x_2 \ldots \alpha_kx_k$ с $\mathrm{inv}(\alpha)$; в этом выражении число a_n отсутствует в $h(\alpha)$.]

b) Обозначьте через f другое однозначное соответствие g, которое имеет два следующих свойства: (i) $\mathrm{ind}(g(\alpha)) = \mathrm{inv}(\alpha)$; (ii) $\mathrm{inv}(g(\alpha)) = \mathrm{ind}(\alpha)$. [*Указание.* Примите во внимание обратные перестановки.]

26. [*M25*] Чему равен статистический коэффициент корреляции между числом инверсий и индексом случайной перестановки? (См. упр. 3.3.2–(24).)

27. [*M37*] Докажите, что в дополнение к (15) существует простое соотношение между $\mathrm{inv}(a_1\,a_2 \ldots a_n)$ и n-мерной строкой $(q_1, q_2, \ldots, q_n)$. Используйте это соотношение для получения производной соотношения (17) в общем случае и формирования алгебраического выражения в виде производящей функции двух переменных

$$H_n(w, z) = \sum w^{\mathrm{inv}(a_1\,a_2 \ldots a_n)} z^{\mathrm{ind}(a_1\,a_2 \ldots a_n)},$$

где суммирование выполняется по всем $n!$ перестановкам $a_1\,a_2 \ldots a_n$.

▶ **28.** [*25*] (Р. В. Флойд (R. W. Floyd), 1983.) Если $a_1\,a_2 \ldots a_n$ — перестановка множества $\{1, 2, \ldots, n\}$, то его *суммарное смещение* определяется как $\sum_{j=1}^{n} |a_j - j|$. Найдите верхнюю и нижнюю границы суммарного смещения в терминах количества инверсий.

29. [*28*] Если $\pi = a_1\,a_2 \ldots a_n$ и $\pi' = a'_1\,a'_2 \ldots a'_n$ суть перестановки множества $\{1, 2, \ldots, n\}$, то обозначим их произведение $\pi\pi'$ как $a'_{a_1}\,a'_{a_2} \ldots a'_{a_n}$. Пусть $\mathrm{inv}(\pi)$ обозначает количество инверсий, как и в упр. 25. Покажите, что $\mathrm{inv}(\pi\pi') \le \mathrm{inv}(\pi) + \mathrm{inv}(\pi')$ и что равенство соблюдается тогда и только тогда, когда $\pi\pi'$ расположено "ниже" π' (как введено в упр. 12).

*5.1.2. Перестановки мультимножества

До сих пор мы рассматривали перестановки *множества* элементов, но это частный случай перестановок *мультимножества*. (Мультимножество — это то же самое, что и множество, но в нем могут содержаться одинаковые элементы. Некоторые основные свойства мультимножеств обсуждались в разделе 4.6.3–19.)

Рассмотрим, например, мультимножество

$$M = \{a, a, a, b, b, c, d, d, d, d\}, \tag{1}$$

в котором содержится 3 элемента a, 2 элемента b, 1 элемент c и 4 элемента d. Повторения элементов в мультимножестве можно записать и другим способом:

$$M = \{3 \cdot a,\ 2 \cdot b,\ c,\ 4 \cdot d\}. \tag{2}$$

Перестановка мультимножества* M — это некоторое расположение его элементов в ряд, например

$$c\ a\ b\ d\ d\ a\ b\ d\ a\ d.$$

С другой стороны, такую последовательность можно назвать буквенной строкой, содержащей 3 буквы a, 2 буквы b, 1 букву c и 4 буквы d. Сколько существует перестановок мультимножества M? Если бы мы рассматривали все элементы M как различные, обозначив их как $a_1, a_2, a_3, b_1, b_2, c_1, d_1, d_2, d_3, d_4$, то получили бы $10! =$ 3,628,800 перестановок, но после отбрасывания индексов многие из них оказались бы одинаковыми. Фактически каждая перестановка M встретилась бы ровно $3!\,2!\,1!\,4! =$ 288 раз, поскольку в любой из них индексы при буквах a_k можно расставить $3!$ способами, при b_k (независимо) — $2!$ способами, при c_k — одним способом, при d_k — $4!$ способами соответственно. Поэтому число перестановок M равно

$$\frac{10!}{3!\,2!\,1!\,4!} = 12{,}600.$$

В применении к общему случаю те же рассуждения показывают, что число перестановок любого мультимножества равно полиномиальному коэффициенту

$$\binom{n}{n_1, n_2, \ldots} = \frac{n!}{n_1!\,n_2!\ldots}, \tag{3}$$

где n_1 — число элементов первого типа, n_2 — число элементов второго типа и т. д., а $n = n_1 + n_2 + \cdots$ — общее число элементов. Количество перестановок множества было известно еще в древние времена. В древнееврейской Книге Творения (ок. 100 г. н. э.), наиболее раннем литературном произведении иудейского философского мистицизма, даны верные значения первых семи факториалов, после чего говорится: "Продолжай, и получишь числа, которые уста не могут произнести, а ухо не может воспринять". [*Сефер Этзирах*, конец части 4. См. Solomon Gandz, *Studies in Hebrew Astronomy and Mathematics* (New York: Ktav, 1970), 494–496; Aryeh Kaplan, *Sefer Yetzirah* (York Beach, Maine: Samuel Weiser, 1993).] Это первый известный в истории подсчет числа перестановок. Второй встречается в индийском классическом произведении *Ануйогадвара-сутра* (ок. 500 г. н. э.), правило 97, в котором приводится формула числа перестановок шести элементов, которые не расположены ни в порядке возрастания, ни в порядке убывания:

$$6 \times 5 \times 4 \times 3 \times 2 \times 1 - 2.$$

[См. G. Chakravarti, *Bull. Calcutta Math. Soc.* **24** (1932), 79–88. "Ануйогадвара-сутра" — одна из книг канонов джайнизма, религиозной секты, распространенной в Индии.] Соответствующее правило для мультимножеств впервые, по-видимому, встречается в книге "Лилавати", написанной Бхаскарой Ахарьей (ок. 1150 г.) (разд. 270–271). У Бхаскары это правило сформулировано весьма сжато и проиллюстрировано лишь двумя простыми примерами: $\{2, 2, 1, 1\}$ и $\{4, 8, 5, 5, 5\}$. В результате в английском переводе это правило не сформулировано корректно, впрочем, имеются некоторые сомнения относительно того, понимал ли сам Бхаскара, о чем говорил.

---

* В англоязычной литературе такие перестановки иногда называют "permatution" в отличие от "permutation".

Вслед за этим правилом Бхаскара приводит интересную формулу

$$\frac{(4+8+5+5+5)\times 120\times 11111}{5\times 6}$$

для суммы 20 чисел $48555 + 45855 + \cdots$. Верное правило для нахождения числа перестановок в случае, когда только один элемент может повторяться, найдено независимо немецким ученым иезуитом Атанасиусом Кирхером (Athanasius Kircher) в его многотомном труде о музыке [*Musurgia Universalis* **2** (Rome: 1650), 5–7]. Кирхера интересовал вопрос о количестве мелодий, которые можно создать из данного набора нот; для этого он придумал то, что называл "музарифметикой". На стр. 18–21 своего труда он дает верное значение числа перестановок мультимножества $\{m\cdot C,\ n\cdot D\}$ при нескольких значениях m и n, хотя описал он свой метод вычислений лишь для случая $n = 1$. Общее правило (3) появилось позже в книге Жана Престэ (J. Prestet) *Elémens de Mathématiques* (Paris: 1675, 351–352), в которой содержится одно из первых изложений комбинаторной математики, написанных в Западной Европе. Престэ верно сформулировал правило для произвольного мультимножества, но проиллюстрировал его лишь простым примером $\{a, a, b, b, c, c\}$. Он особо отметил, что деление на *сумму* факториалов, которое он считал естественным обобщением правила Кирхера, было бы ошибкой. Несколько лет спустя Джон Валлис (John Wallis) в своей книге *Discourse of Combinations* (Oxford: 1685, Chapter 2), опубликованной вместе с его же *Treatise of Algebra*, рассмотрел это правило более подробно.

В 1965 году Доминик Фоата (Dominique Foata) ввел одно интересное понятие, так называемое "соединительное произведение" (intercalation product), которое позволило распространить многие известные результаты, касающиеся обычных перестановок, на общий случай перестановок мультимножества. [См. *Publ. Inst. Statistique*, Univ. Paris, **14** (1965), 81–241; а также *Lecture Notes in Math.* **85** (Springer, 1969).] Предполагая, что элементы мультимножества каким-то способом линейно упорядочены, можно рассмотреть *двухстрочную нотацию*, например

$$\begin{pmatrix} a & a & a & b & b & c & d & d & d & d \\ c & a & b & d & d & a & b & d & a & d \end{pmatrix}. \tag{4}$$

Здесь верхняя строка содержит элементы M в порядке неубывания и нижняя — это сама перестановка. *Соединительное произведение* $\alpha\top\beta$ двух перестановок мультимножеств α и β — это перестановка, которая получается, если (a) взять двухстрочные обозначения для α и β, (b) записать соответствующие строки в одну и (c) рассортировать столбцы так, чтобы элементы верхней строки расположились в порядке неубывания. Сортировка должна быть устойчивой в том смысле, что взаимное расположение элементов нижней строки сохраняется, если соответствующие элементы верхней строки равны. Например, $c\ a\ d\ a\ b \top b\ d\ d\ a\ d = c\ a\ b\ d\ d\ a\ b\ d\ a\ d$, так как

$$\begin{pmatrix} a & a & b & c & d \\ c & a & d & a & b \end{pmatrix} \top \begin{pmatrix} a & b & d & d & d \\ b & d & d & a & d \end{pmatrix} = \begin{pmatrix} a & a & a & b & b & c & d & d & d & d \\ c & a & b & d & d & a & b & d & a & d \end{pmatrix}. \tag{5}$$

Нетрудно видеть, что операция соединительного произведения ассоциативна, т. е.

$$(\alpha\top\beta)\top\gamma = \alpha\top(\beta\top\gamma), \tag{6}$$

и что она подчиняется законам сокращения

$$\begin{aligned} \pi \top \alpha = \pi \top \beta \quad &\text{влечет за собой} \quad \alpha = \beta, \\ \alpha \top \pi = \beta \top \pi \quad &\text{влечет за собой} \quad \alpha = \beta. \end{aligned} \tag{7}$$

Существует "единичный элемент"

$$\alpha \top \epsilon = \epsilon \top \alpha = \alpha, \tag{8}$$

где ϵ — нуль-перестановка, "расположение в ряд" элементов пустого множества. Закон коммутативности, вообще говоря, не выполняется (см. упр. 2), тем не менее

$$\alpha \top \beta = \beta \top \alpha, \qquad \text{если } \alpha \text{ и } \beta \text{ не содержат общих букв.} \tag{9}$$

Аналогичным способом понятие *цикла* можно распространить на случай, когда элементы могут повторяться. Будем записывать в виде

$$(x_1 \quad x_2 \quad \ldots \quad x_n) \tag{10}$$

перестановку, двухстрочное представление которой получается путем устойчивой сортировки столбцов

$$\begin{pmatrix} x_1 & x_2 & \ldots & x_n \\ x_2 & x_3 & \ldots & x_1 \end{pmatrix} \tag{11}$$

по верхним элементам. Например,

$$(d\ b\ d\ d\ a\ c\ a\ a\ b\ d) = \begin{pmatrix} d & b & d & d & a & c & a & a & b & d \\ b & d & d & a & c & a & a & b & d & d \end{pmatrix} = \begin{pmatrix} a & a & a & b & b & c & d & d & d & d \\ c & a & b & d & d & a & b & d & a & d \end{pmatrix},$$

так что перестановка (4) фактически является циклом. Мы могли бы описать этот цикл словесно, сказав что-нибудь наподобие "d переходит в b, переходит в d, переходит в d, переходит в ... переходит в d и возвращается обратно". Заметим, что эти обобщенные циклы не обладают всеми свойствами обычных циклов: $(x_1\, x_2 \ldots x_n)$ — необязательно то же самое, что и $(x_2 \ldots x_n\, x_1)$. В разделе 1.3.3 мы выяснили, что каждую перестановку множества можно единственным с точностью до порядка сомножителей образом представить в виде произведения непересекающихся циклов, где "произведение" перестановок определяется законом композиции. Легко видеть, что *произведение непересекающихся циклов — то же самое, что их соединительное произведение*; это наводит на мысль о том, что можно будет обобщить полученные ранее результаты, если найти единственное (в каком-то смысле) представление и для произвольной перестановки мультимножества в виде соединительного произведения циклов. В действительности существует, по крайней мере, два естественных способа сделать это, и каждый из них имеет важные приложения. Равенство (5) дает один способ представления $c\ a\ b\ d\ d\ a\ b\ d\ a\ d$ в виде соединительного произведения более коротких перестановок. Рассмотрим общую задачу о нахождении всех разложений $\pi = \alpha \top \beta$ данной перестановки π. Для этого полезно проанализировать конкретную перестановку, например

$$\pi = \begin{pmatrix} a & a & b & b & b & b & b & c & c & c & d & d & d & d & d \\ d & b & c & b & c & a & c & d & a & d & d & b & b & b & d \end{pmatrix}. \tag{12}$$

Если можно записать π в виде $\alpha \top \beta$, где α содержит, по крайней мере, одну букву a, то крайнее слева a в верхней строке двухстрочного представления α должно

оказаться над d; значит, перестановка α должна содержать хотя бы одну букву d. Если взглянуть теперь на крайнее слева d в верхней строке α, то можно точно так же увидеть, что оно должно оказаться над d; значит, в α должны содержаться, по меньшей мере, *две* буквы d. Посмотрев на второе d, видим, что α содержит также b. Единственное предположение о том, что α есть левый сомножитель π, содержащий букву a, приводит к такому промежуточному результату:

$$\alpha = \begin{pmatrix} a & \cdots & b & \cdots & d & d & \cdots \\ d & & & & d & b & \end{pmatrix}. \tag{13}$$

Продолжая рассуждать точно так же, обнаружим, что буква b в верхней строке (13) должна оказаться над c, и т. д. В конце концов, этот процесс вновь приведет нас к букве a, и мы сможем, если захотим, отождествить ее с первой буквой a. Такое рассуждение, по существу, доказывает, что любой левый сомножитель α в разложении перестановки (12), содержащий a, имеет вид $(d\ d\ b\ c\ d\ b\ b\ c\ a) \top \alpha'$, где α' — некоторая перестановка. (Удобно записывать a не в начале, а в конце цикла; это допустимо, поскольку буква a только одна.) Аналогично, если бы мы предположили, что α содержит букву b, то вывели бы, что $\alpha = (c\ d\ d\ b) \top \alpha''$, где α'' — некоторая перестановка. В общем случае эти рассуждения показывают, что *если есть какое-нибудь разложение $\alpha \top \beta = \pi$, где α содержит данную букву y, то существует единственный цикл вида*

$$(x_1\ \ldots\ x_n\ y), \qquad n \geq 0, \qquad x_1, \ldots, x_n \neq y, \tag{14}$$

который является левым сомножителем в разложении перестановки α. Такой цикл легко отыскать, зная π и y; это самый короткий левый сомножитель в разложении перестановки π, содержащий букву y. Из сказанного следует теорема.

Теорема А. *Пусть элементы мультимножества M линейно упорядочены отношением "<". Каждая перестановка π мультимножества M имеет единственное представление в виде соединительного произведения*

$$\pi = (x_{11} \ldots x_{1n_1} y_1) \top (x_{21} \ldots x_{2n_2} y_2) \top \cdots \top (x_{t1} \ldots x_{tn_t} y_t), \quad t \geq 0, \tag{15}$$

удовлетворяющее следующим двум условиям:

$$y_1 \leq y_2 \leq \cdots \leq y_t \qquad \text{и} \qquad y_i < x_{ij} \quad \text{при } 1 \leq j \leq n_i,\ 1 \leq i \leq t. \tag{16}$$

(Иными словами, в каждом цикле последний элемент меньше любого другого и последние элементы циклов образуют неубывающую последовательность.)

Доказательство. При $\pi = \epsilon$ получим требуемое разложение, положив $t = 0$. В противном случае пусть y_1 — минимальный элемент в π; определим $(x_{11} \ldots x_{1n_1} y_1)$ — самый короткий левый сомножитель разложения π, содержащий y_1, как в рассмотренном примере. Теперь $\pi = (x_{11}\ \ldots\ x_{1n_1}\ y_1) \top \rho$, где ρ — некоторая перестановка; применив индукцию по длине перестановки, можем написать

$$\rho = (x_{21}\ \ldots\ x_{2n_2}\ y_2) \top \cdots \top (x_{t1}\ \ldots\ x_{tn_t}\ y_t), \quad t \geq 1,$$

где условия (16) выполнены. Тем самым доказано существование такого разложения.

Докажем единственность разложения (15), удовлетворяющего условиям (16). Ясно, что $t = 0$ тогда и только тогда, когда π — нуль-перестановка ϵ. При $t > 0$ из (16) следует, что y_1 — минимальный элемент перестановки и что $(x_{11} \ldots x_{1n_1} y_1)$ — самый короткий левый сомножитель, содержащий y_1. Поэтому $(x_{11} \ldots x_{1n_1} y_1)$ определяется однозначно; доказательство единственности такого представления завершается применением индукции и законов сокращения (7). ▮

Например, "каноническое" разложение перестановки (12), удовлетворяющее данным условиям, таково:

$$(d\ d\ b\ c\ d\ b\ b\ c\ a)_{\top}(b\ a)_{\top}(c\ d\ b)_{\top}(d), \tag{17}$$

если $a < b < c < d$.

Важно отметить, что на самом деле в этом определении можно отбросить скобки и знаки операции $_{\top}$ и это не приведет к неоднозначности! В конце каждого цикла появляется наименьший из оставшихся элементов. Таким образом, наше построение связывает с исходной перестановкой

$$\pi = d\ b\ c\ b\ c\ a\ c\ d\ a\ d\ d\ b\ b\ b\ d$$

перестановку

$$\pi' = d\ d\ b\ c\ d\ b\ b\ c\ a\ b\ a\ c\ d\ b\ d.$$

Если в двухстрочном представлении π содержится столбец вида $\frac{y}{x}$, где $x < y$, то в связанной с π' перестановке присутствует соответствующая пара соседних элементов $\ldots y\, x \ldots$. Так, в нашем примере π содержит три столбца вида $\frac{d}{b}$, трижды встречается пара $d\,b$. Вообще, из этого построения вытекает замечательная теорема.

Теорема B. *Пусть M — мультимножество. Существует взаимно однозначное соответствие между перестановками M, такое, что если π соответствует π', то выполняются следующие условия:*

a) *крайний слева элемент π' равен крайнему слева элементу π;*

b) *для всех пар участвующих в перестановке элементов (x, y), таких, что $x < y$, число вхождений столбца $\frac{y}{x}$ в двухстрочное представление перестановки π равно числу случаев, когда в перестановке π' элемент x следует непосредственно за y.* ▮

Если M — множество, то это, по существу, "нестандартное соответствие", которое обсуждалось в конце раздела 1.3.3, с незначительными изменениями. Более общий результат теоремы B полезен при подсчете числа перестановок специальных типов, поскольку часто проще решить задачу с ограничениями, наложенными на двухстрочное представление, чем эквивалентную задачу с ограничениями на пары соседних элементов.

П. А. Мак-Магон (P. A. MacMahon) рассмотрел задачи этого типа в своей выдающейся книге *Combinatory Analysis* **1** (Cambridge Univ. Press, 1915), 168–186. Он дал конструктивное доказательство теоремы B в частном случае, когда M содержит элементы лишь двух различных типов, скажем a и b; его построение для этого случая, по существу, совпадает с приведенным здесь, но представлено в совершенно ином виде. Для случая трех различных элементов a, b, c Мак-Магон дал сложное неконструктивное доказательство теоремы B; общий случай впервые доказал Фоата (Foata) [см. *Comptes Rendus Acad. Sci. Paris* **258** (1964), 1672–1675].

В качестве нетривиального примера применения теоремы B найдем число строк букв a, b, c, содержащих ровно

$$\begin{array}{ll} A & \text{вхождений буквы } a; \\ B & \text{вхождений буквы } b; \\ C & \text{вхождений буквы } c; \\ k & \text{вхождений пары стоящих рядом букв } ca; \\ l & \text{вхождений пары стоящих рядом букв } cb; \\ m & \text{вхождений пары стоящих рядом букв } ba. \end{array} \tag{18}$$

Из теоремы следует, что это то же самое, что найти число двухстрочных массивов вида

$$\begin{array}{c} \overbrace{\qquad\qquad}^{A} \quad \overbrace{\qquad\qquad}^{B} \quad \overbrace{\qquad\qquad}^{C} \\ \begin{pmatrix} a & \dots & a & b & \dots & b & c & \dots & c \\ \sqcup & \dots & \sqcup & \sqcup & \dots & \sqcup & \sqcup & \dots & \sqcup \end{pmatrix} \\ \underbrace{\qquad\qquad}_{A-k-m\,a\text{'s}} \quad \underbrace{\qquad\qquad}_{m\ a\text{'s}} \quad \underbrace{\qquad\qquad}_{k\ a\text{'s}} \\ \underbrace{\qquad\qquad\qquad\qquad}_{B-l\ b\text{'s}} \quad \underbrace{\qquad\qquad}_{l\ b\text{'s}} \\ \underbrace{\qquad\qquad\qquad\qquad\qquad\qquad}_{C\,c\text{'s}} \end{array} \tag{19}$$

Буквы a можно расположить во второй строке

$$\binom{A}{A-k-m}\binom{B}{m}\binom{C}{k} \qquad \text{способами;}$$

после этого буквы b можно разместить в оставшихся позициях

$$\binom{B+k}{B-l}\binom{C-k}{l} \qquad \text{способами.}$$

Остальные свободные места нужно заполнить буквами c; следовательно, искомое число равно

$$\binom{A}{A-k-m}\binom{B}{m}\binom{C}{k}\binom{B+k}{B-l}\binom{C-k}{l}. \tag{20}$$

Вернемся к вопросу о нахождении всех разложений данной перестановки. Существует ли такой объект, как "простая" перестановка, которая не разлагается на множители, отличные от нее самой и ϵ? Обсуждение, предшествующее теореме A, немедленно приводит к выводу о том, что *перестановка будет простой тогда и только тогда, когда она является циклом без повторяющихся элементов.* В случае, если перестановка является таким циклом, наше рассуждение доказывает, что не существует левых множителей, кроме ϵ и самого цикла. Если же перестановка содержит повторяющийся элемент y, то всегда можно выделить нетривиальный цикл в качестве левого сомножителя, в котором элемент y встречается всего лишь однажды.

Если перестановка непростая, то ее можно разлагать на все меньшие и меньшие части, пока не будет получено произведение простых перестановок. Можно даже показать, что такое разложение является единственным с точностью до порядка записи коммутирующих сомножителей.

Теорема C. *Каждую перестановку мультимножества можно записать в виде произведения*

$$\sigma_1 \top \sigma_2 \top \cdots \top \sigma_t, \qquad t \geq 0, \tag{21}$$

где σ_j — циклы, не содержащие повторяющихся элементов. Это представление единственное в том смысле, что два любых таких представления одной и той же перестановки можно преобразовать одно в другое, последовательно меняя местами соседние непересекающиеся циклы.

Термин "непересекающиеся циклы" относится к циклам, не имеющим общих элементов. В качестве примера можно проверить, что перестановка

$$\begin{pmatrix} a & a & b & b & c & c & d \\ b & a & a & c & d & b & c \end{pmatrix}$$

разлагается на множители ровно пятью способами:

$$\begin{aligned}(a\ b) \top (a) \top (c\ d) \top (b\ c) &= (a\ b) \top (c\ d) \top (a) \top (b\ c) \\ &= (a\ b) \top (c\ d) \top (b\ c) \top (a) \\ &= (c\ d) \top (a\ b) \top (b\ c) \top (a) \\ &= (c\ d) \top (a\ b) \top (a) \top (b\ c).\end{aligned} \tag{22}$$

Доказательство. Нужно установить, что выполняется сформулированное в теореме свойство единственности. Применим индукцию по длине перестановки; тогда достаточно доказать, что если ρ и σ — два различных цикла, не содержащих повторяющихся элементов, и

$$\rho \top \alpha = \sigma \top \beta,$$

то ρ и σ — непересекающиеся циклы и

$$\alpha = \sigma \top \theta, \qquad \beta = \rho \top \theta,$$

где θ — некоторая перестановка. Пусть y — произвольный элемент цикла ρ, тогда у любого левого сомножителя в разложении $\sigma \top \beta$, содержащего этот элемент y, будет левый сомножитель ρ. Значит, если ρ и σ имеют общий элемент, цикл σ должен быть кратен ρ; тогда $\sigma = \rho$ (так как они простые), что противоречит нашему предположению. Следовательно, цикл, содержащий y и не имеющий общих элементов с σ, должен быть левым сомножителем в разложении β. Применив законы сокращения (7), завершим доказательство. ∎

В качестве иллюстрации теоремы C рассмотрим перестановки мультимножества $M = \{A \cdot a,\ B \cdot b,\ C \cdot c\}$, состоящего из A элементов a, B элементов b и C элементов c. Пусть $N(A, B, C, m)$ — число перестановок мультимножества M, двухстрочное представление которых *не содержит* столбцов вида $\binom{a}{a}$, $\binom{b}{b}$, $\binom{c}{c}$ и содержит ровно m столбцов вида $\binom{a}{b}$. Отсюда следует, что имеется ровно $A - m$ столбцов вида $\binom{a}{c}$, $B - m$ столбцов вида $\binom{c}{b}$, $C - B + m$ столбцов вида $\binom{c}{a}$, $C - A + m$ столбцов вида $\binom{b}{c}$ и $A + B - C - m$ столбцов вида $\binom{b}{a}$; следовательно,

$$N(A, B, C, m) = \binom{A}{m} \binom{B}{C - A + m} \binom{C}{B - m}. \tag{23}$$

Теорема C предлагает другой способ подсчета этих перестановок: коль скоро столбцы $\begin{smallmatrix}a\\a\end{smallmatrix}$, $\begin{smallmatrix}b\\b\end{smallmatrix}$, $\begin{smallmatrix}c\\c\end{smallmatrix}$ исключены, в разложении перестановки единственно возможны такие простые множители:

$$(a\ b),\qquad (a\ c),\qquad (b\ c),\qquad (a\ b\ c),\qquad (a\ c\ b). \tag{24}$$

Каждая пара этих циклов имеет хотя бы одну общую букву, значит, разложение единственно. Если цикл $(a\ b\ c)$ встречается в разложении k раз, то из нашего предыдущего предположения следует, что $(a\ b)$ встречается $m-k$ раз, $(b\ c)$ встречается $C-A+m-k$ раз, $(a\ c)$ встречается $C-B+m-k$ раз и $(a\ c\ b)$ встречается $A+B-C-2m+k$ раз. Следовательно, $N(A,B,C,m)$ равно числу перестановок этих циклов (полиномиальному коэффициенту), просуммированному по всем значениям k:

$$\begin{aligned} &N(A,B,C,m)\\ &\quad=\sum_k \frac{(C+m-k)!}{(m-k)!\,(C-A+m-k)!\,(C-B+m-k)!\,k!\,(A+B-C-2m+k)!}\\ &\quad=\sum_k \binom{m}{k}\binom{A}{m}\binom{A-m}{C-B+m-k}\binom{C+m-k}{A}. \end{aligned} \tag{25}$$

Сравнивая (25) с (23), обнаруживаем, что должно выполняться тождество

$$\sum_k \binom{m}{k}\binom{A-m}{C-B+m-k}\binom{C+m-k}{A} = \binom{B}{C-A+m}\binom{C}{B-m}. \tag{26}$$

Оказывается, с этим тождеством мы встречались в упр. 1.2.6–31:

$$\sum_j \binom{M-R+S}{j}\binom{N+R-S}{N-j}\binom{R+j}{M+N} = \binom{R}{M}\binom{S}{N}, \tag{27}$$

где $M=A+B-C-m$, $N=C-B+m$, $R=B$, $S=C$, а $j=C-B+m-k$.

Аналогично можно подсчитать число перестановок мультимножества $\{A\cdot a, B\cdot b, C\cdot c, D\cdot d\}$, если количество столбцов различных типов в них задано следующим образом.

Тип столбца:	$\begin{smallmatrix}a\\d\end{smallmatrix}$	$\begin{smallmatrix}a\\b\end{smallmatrix}$	$\begin{smallmatrix}b\\a\end{smallmatrix}$	$\begin{smallmatrix}b\\c\end{smallmatrix}$	$\begin{smallmatrix}c\\b\end{smallmatrix}$	$\begin{smallmatrix}c\\d\end{smallmatrix}$	$\begin{smallmatrix}d\\a\end{smallmatrix}$	$\begin{smallmatrix}d\\c\end{smallmatrix}$
Частота:	r	$A-r$	q	$B-q$	$B-A+r$	$D-r$	$A-q$	$D-A+q$

(28)

(Здесь $A+C=B+D$.) Возможными циклами в разложении такой перестановки на простые множители будут

Цикл:	$(a\ b)$	$(b\ c)$	$(c\ d)$	$(d\ a)$	$(a\ b\ c\ d)$	$(d\ c\ b\ a)$
Частота:	$A-r-s$	$B-q-s$	$D-r-s$	$A-q-s$	s	$q-A+r+s$

(29)

при некотором s (см. упр. 12). В этом случае циклы $(a\ b)$ и $(c\ d)$ заменяют друг друга так же, как циклы $(b\ c)$ и $(d\ a)$, поэтому необходимо подсчитать число различных разложений на простые множители. Оказывается (см. упр. 10), всегда существует единственное разложение, такое, что цикл $(a\ b)$ никогда не следует непосредственно

за (с d), а (b c) не встречается сразу после (d a). Отсюда, используя результат упр. 13, получаем тождество

$$\sum_{s,t}\binom{B}{t}\binom{A-q-s}{A-r-s-t}\binom{B+D-r-s-t}{B-q-s}$$
$$\times\frac{D!}{(D-r-s)!\,(A-q-s)!\,s!\,(q-A+r+s)!}$$
$$=\binom{A}{r}\binom{B+D-A}{D-r}\binom{B}{q}\binom{D}{A-q}.$$

Вынося из обеих частей множитель $\binom{D}{A-q}$ и несколько упрощая факториалы, приходим к сложному на вид пятипараметрическому тождеству биномиальных коэффициентов:

$$\sum_{s,t}\binom{B}{t}\binom{A-r-t}{s}\binom{B+D-r-s-t}{D+q-r-t}\binom{D-A+q}{D-r-s}\binom{A-q}{r+t-q}$$
$$=\binom{A}{r}\binom{B+D-A}{D-r}\binom{B}{q}. \qquad (30)$$

Используя тождество (27), можно выполнить суммирование по s, а затем легко вычислить получившуюся сумму по t. Таким образом, выполнив всю работу, мы не смогли обнаружить какое-либо тождество, которое мы бы еще не умели выводить. Но мы, по крайней мере, научились подсчитывать число перестановок определенного вида двумя различными способами, а эти методы подсчета — хорошая подготовка к решению задач, которые еще ждут нас впереди.

УПРАЖНЕНИЯ

1. [*M05*] *Да или нет?* Пусть M_1 и M_2 — мультимножества. Если α — перестановка M_1, а β — перестановка M_2, то $\alpha\top\beta$ — перестановка $M_1\cup M_2$.

2. [*10*] Соединительное произведение перестановок $c\ a\ d\ a\ b$ и $b\ d\ d\ a\ d$ представлено в (5); найдите соединительное произведение $b\ d\ d\ a\ d\ \top\ c\ a\ d\ a\ b$, которое получается, если сомножители поменять местами.

3. [*M13*] Верно ли утверждение, обратное (9)? Иначе говоря, если перестановки α и β коммутативны относительно операции соединительного произведения, то следует ли из этого, что они не содержат общих букв?

4. [*M11*] Каноническое разложение перестановки (12) в соответствии с теоремой А при $a<b<c<d$ задается формулой (17). Найдите каноническое разложение в случае, когда $d<c<b<a$.

5. [*M23*] В условии (b) теоремы В требуется, чтобы x было меньше y. Что будет, если ослабить это требование, заменив его требованием $x\le y$?

6. [*M15*] Сколько существует цепочек, состоящих ровно из m букв a и n букв b, таких, что ровно k букв b стоят непосредственно перед буквами a и нет никаких других букв, кроме a и b?

7. [*M21*] Сколько строк из букв a, b, с, удовлетворяющих условиям (18), начинаются с буквы a, с буквы b, с буквы c?

▸ **8.** [*20*] Найдите все разложения перестановки (12) на два множителя $\alpha\top\beta$.

9. [*33*] Напишите программы, которые формировали бы разложения заданной перестановки мультимножества, описанные в теоремах A и C.

▶ **10.** [*M30*] *Да или нет?* Согласно теореме C разложение на простые множители не вполне однозначно, тем не менее можно следующим образом обеспечить единственность. "Существует линейное упорядочение $\prec$ на множестве простых перестановок, такое, что каждая перестановка мультимножества имеет единственное разложение $\sigma_1 \top \sigma_2 \top \cdots \top \sigma_n$ на простые множители, удовлетворяющее условию $\sigma_i \preceq \sigma_{i+1}$, если σ_i коммутирует с σ_{i+1} при $1 \le i < n$".

▶ **11.** [*M26*] Пусть $\sigma_1, \sigma_2, \ldots, \sigma_t$ — циклы без повторяющихся элементов. Определим частичное упорядочение $\prec$ на множестве t элементов $\{x_1, \ldots, x_t\}$, полагая $x_i \prec x_j$, если $i < j$ и σ_i имеет, по крайней мере, одну общую букву с σ_j. Докажите следующую связь между теоремой C и понятием "топологическая сортировка" (раздел 2.2.3): *число различных разложений перестановки $\sigma_1 \top \sigma_2 \top \cdots \top \sigma_t$ на простые множители равно количеству способов топологической сортировки данного частичного упорядочения*. (Например, в соответствии с (22) существует пять способов топологической сортировки упорядочения $x_1 \prec x_2$, $x_3 \prec x_4$, $x_1 \prec x_4$.) Обратно, если на множестве из t элементов задано какое-либо частичное упорядочение, то существует множество циклов $\{\sigma_1, \sigma_2, \ldots, \sigma_t\}$, которое определяет это частичное упорядочение указанным способом.

12. [*M16*] Покажите, что (29) есть следствие, вытекающее из предположения (28).

13. [*M21*] Докажите, что число перестановок мультимножества

$$\{A \cdot a,\ B \cdot b,\ C \cdot c,\ D \cdot d,\ E \cdot e,\ F \cdot f\},$$

не содержащих пар стоящих рядом букв ca и db, равно

$$\sum_t \binom{D}{A-t}\binom{A+B+E+F}{t}\binom{A+B+C+E+F-t}{B}\binom{C+D+E+F}{C,D,E,F}.$$

14. [*M30*] Один из способов определить перестановку π^-, обратную перестановке π, который подсказан нам в других определениях этого раздела, — поменять местами строки двухстрочного представления π и так выполнить устойчивую сортировку столбцов, чтобы элементы верхней строки расположились в порядке неубывания. Например, если $a < b < c < d$, то из этого определения следует, что обратной перестановкой к $c\ a\ b\ d\ d\ a\ b\ d\ a\ d$ будет $a\ c\ d\ a\ d\ a\ b\ b\ d\ d$.

Исследуйте свойства этой операции обращения; имеется ли, например, какая-нибудь простая связь между данной операцией и соединительным произведением? Можно ли подсчитать число перестановок, таких, что $\pi = \pi^-$?

▶ **15.** [*M25*] Докажите, что перестановка $a_1 \ldots a_n$ мультимножества

$$\{n_1 \cdot x_1,\ n_2 \cdot x_2, \ldots, n_m \cdot x_m\},$$

где $x_1 < x_2 < \cdots < x_m$ и $n_1 + n_2 + \cdots + n_m = m$, является циклом тогда и только тогда, когда ориентированный граф с вершинами $\{x_1, x_2, \ldots, x_m\}$ и дугами из x_j в $a_{n_1+\cdots+n_j}$ содержит ровно один ориентированный цикл. В таком случае число способов представления перестановки в виде цикла равно длине этого ориентированного цикла. Например, ориентированным графом, соответствующим перестановке

$$\begin{pmatrix} a & a & a & b & b & c & c & c & d & d \\ d & c & b & a & c & a & a & b & d & c \end{pmatrix}, \qquad \text{будет} \qquad \begin{tikzcd} a \arrow[r] & b \arrow[d, bend left] \\ d \arrow[r] & c \arrow[u, bend left] \end{tikzcd},$$

а два способа представления перестановки в виде цикла имеют вид $(b\ a\ d\ d\ c\ a\ c\ a\ b\ c)$ и $(c\ a\ d\ d\ c\ a\ c\ b\ a\ b)$.

16. [*M35*] В предыдущем разделе, формула 5.1.1–(8), мы нашли производящую функцию для *инверсий* перестановок в частном случае, когда в перестановке участвуют элементы множества. Покажите, что в общем случае перестановок *мультимножества* производящая функция для инверсий $\{n_1 \cdot x_1, n_2 \cdot x_2, \ldots\}$ равна "z-полиномиальному коэффициенту"

$$\binom{n}{n_1, n_2, \ldots}_z = \frac{n!_z}{n_1!_z\, n_2!_z \ldots}, \qquad \text{где} \quad m!_z = \prod_{k=1}^{m} (1 + z + \cdots + z^{k-1}).$$

[Ср. с (3) и с определением z-номиальных коэффициентов в формуле 1.2.6–(40).]

17. [*M24*] С помощью производящей функции, найденной в упр. 16, вычислите среднее значение и дисперсию для числа инверсий в случайной перестановке мультимножества.

18. [*M30*] (П. А. Мак-Магон.) *Индекс* перестановки $a_1\, a_2 \ldots a_n$ был определен в предыдущем упражнении; мы доказали, что число перестановок этого множества, имеющих данный индекс k, равно числу перестановок, имеющих k инверсий. Верно ли это для перестановок заданного мультимножества?

19. [*HM28*] Определим *функцию Мёбиуса* $\mu(\pi)$ перестановки π: она равна 0, если π содержит повторяющиеся элементы, и $(-1)^k$ в противном случае, если π — произведение k простых перестановок. (Ср. с определением обычной функции Мёбиуса; упр. 4.5.2–10.)

a) Докажите, что если $\pi \neq \epsilon$, то

$$\sum \mu(\lambda) = 0,$$

где сумма берется по всем перестановкам λ, являющимся левыми сомножителями в разложении π (т. е. по всем λ, таким, что $\pi = \lambda \top \rho$, где ρ — некоторая перестановка).

b) Докажите, что если $x_1 < x_2 < \cdots < x_m$ и $\pi = x_{j_1}\, x_{j_2} \ldots x_{j_n}$, где $1 \le j_k \le m$ при $1 \le k \le n$, то

$$\mu(\pi) = (-1)^n \epsilon(i_1\, i_2 \ldots i_n), \qquad \text{где} \quad \epsilon(i_1\, i_2 \ldots i_n) = \operatorname{sign} \prod_{1 \le j < k \le n} (i_k - i_j).$$

▶ **20.** [*HM33*] (Д. Фоата.) Пусть (a_{ij}) — произвольная матрица действительных чисел. Пользуясь обозначениями из упр. 19, (b), определим $\nu(\pi) = a_{i_1 j_1} \ldots a_{i_n j_n}$, где двухстрочное представление перестановки π таково:

$$\begin{pmatrix} x_{i_1} & x_{i_2} & \ldots & x_{i_n} \\ x_{j_1} & x_{j_2} & \ldots & x_{j_n} \end{pmatrix}.$$

Эта функция полезна при вычислении производящих функций для перестановок мультимножества, потому что $\sum \nu(\pi)$, где сумма берется по всем перестановкам π мультимножества

$$\{n_1 \cdot x_1, \ldots, n_m \cdot x_m\},$$

будет производящей функцией для числа перестановок, удовлетворяющих некоторым ограничениям. Например, если положить $a_{ij} = z$ при $i = j$ и $a_{ij} = 1$ при $i \neq j$, то $\sum \nu(\pi)$ — производящая функция для числа "неподвижных точек" (столбцов, в которых верхний и нижний элементы равны). Чтобы можно было исследовать $\sum \nu(\pi)$ для всех мультимножеств одновременно, рассмотрим функцию

$$G = \sum \pi \nu(\pi),$$

где сумма берется по всем π из множества $\{x_1, \ldots, x_m\}^*$ всех перестановок мультимножеств, содержащих элементы $x_1, \ldots, x_m$, и посмотрим на коэффициенты при $x_1^{n_1} \ldots x_m^{n_m}$ в G.

В этой формуле для G будем рассматривать π как произведение переменных x. Например, при $m = 2$ имеем

$$\begin{aligned} G &= 1+x_1\nu(x_1)+x_2\nu(x_2)+x_1x_1\nu(x_1x_1)+x_1x_2\nu(x_1x_2)+x_2x_1\nu(x_2x_1)+x_2x_2\nu(x_2x_2)+\cdots \\ &= 1+x_1a_{11}+x_2a_{22}+x_1^2a_{11}^2+x_1x_2a_{11}a_{22}+x_1x_2a_{21}a_{12}+x_2^2a_{22}^2+\cdots. \end{aligned}$$

Таким образом, коэффициент при $x_1^{n_1}\ldots x_m^{n_m}$ в G равен сумме $\sum\nu(\pi)$ по всем перестановкам π мультимножества $\{n_1\cdot x_1,\ldots,n_m\cdot x_m\}$. Нетрудно видеть, что этот коэффициент равен также коэффициенту при $x_1^{n_1}\ldots x_m^{n_m}$ в выражении

$$(a_{11}x_1+\cdots+a_{1m}x_m)^{n_1}(a_{21}x_1+\cdots+a_{2m}x_m)^{n_2}\ldots(a_{m1}x_1+\cdots+a_{mm}x_m)^{n_m}.$$

Цель данного упражнения — доказать формулу

$$G = 1/D, \quad \text{где} \quad D = \det\begin{pmatrix} 1-a_{11}x_1 & -a_{12}x_2 & \ldots & -a_{1m}x_m \\ -a_{21}x_1 & 1-a_{22}x_2 & & -a_{2m}x_m \\ \vdots & & & \vdots \\ -a_{m1}x_1 & -a_{m2}x_2 & \ldots & 1-a_{mm}x_m \end{pmatrix},$$

которую П. А. Мак-Магон (P. A. MacMahon) в своей книге *Combinatory Analysis* **1** (1915), разд. 3, назвал "основной теоремой". Например, если $a_{ij} = 1$ при всех i и j, то эта формула дает

$$G = 1/(1-(x_1+x_2+\cdots+x_m))$$

и коэффициент при $x_1^{n_1}\ldots x_m^{n_m}$ оказывается равным $(n_1+\cdots+n_m)!/n_1!\ldots n_m!$, как и должно быть. Для доказательства основной теоремы Мак-Магона покажите, что

a) $\nu(\pi\top\rho) = \nu(\pi)\nu(\rho)$;

b) в обозначениях из упр. 19 $D = \sum\pi\mu(\pi)\nu(\pi)$, где сумма берется по всем перестановкам π из множества $\{x_1,\ldots,x_m\}^*$;

c) из (a) и (b) следует $D\cdot G = 1$.

21. [*M21*] Пусть заданы $n_1,\ldots,n_m$ и $d\ge 0$. Сколько перестановок $a_1a_2\ldots a_n$ мультимножества $\{n_1\cdot 1,\ldots,n_m\cdot m\}$ удовлетворяют условию $a_{j+1}\ge a_j-d$ при $1\le j<n=n_1+\cdots+n_m$?

22. [*M30*] Пусть $P(x_1^{n_1}\ldots x_m^{n_m})$ означает множество всех возможных перестановок мультимножества $\{n_1\cdot x_1,\ldots,n_m\cdot x_m\}$ и пусть $P_0(x_0^{n_0}x_1^{n_1}\ldots x_m^{n_m})$ — подмножество $P(x_0^{n_0}x_1^{n_1}\ldots x_m^{n_m})$, в котором первые n_0 элементов не равны x_0.

a) Задав число t, $1\le t<m$, найдите однозначное соответствие между $P(1^{n_1}\ldots m^{n_m})$ и множеством всех упорядоченных пар перестановок, которые соответственно принадлежат $P_0(0^k1^{n_1}\ldots t^{n_t})$ и $P_0(0^k(t+1)^{n_{t+1}}\ldots m^{n_m})$, для некоторого $k\ge 0$. [*Указание.* Для каждого $\pi = a_1\ldots a_n\in P(1^{n_1}\ldots m^{n_m})$ положите, что $l(\pi)$ — перестановка, полученная путем замены $t+1,\ldots,m$ значением 0 и удаления всех 0 на последних $n_{t+1}+\cdots+n_m$ позициях; аналогично положите, что $r(\pi)$ — перестановка, полученная путем замены $1,\ldots,t$ значением 0 и удаления всех 0 на первых $n_1+\cdots+n_t$ позициях.]

b) Докажите, что число перестановок $P_0(0^{n_0}1^{n_1}\ldots m^{n_m})$, имеющих в двухстрочном представлении p_j столбцов $\binom{0}{j}$ и q_j столбцов $\binom{j}{0}$, равно

$$\frac{|P(x_1^{p_1}\ldots x_m^{p_m}y_1^{n_1-p_1}\ldots y_m^{n_m-p_m})|\,|P(x_1^{q_1}\ldots x_m^{q_m}y_1^{n_1-q_1}\ldots y_m^{n_m-q_m})|}{|P_0(0^{n_0}1^{n_1}\ldots m^{n_m})|}.$$

c) Пусть $w_1,\ldots,w_m$, $z_1,\ldots,z_m$ — комплексные числа на единичной окружности. Определите вес $w(\pi)$ перестановки $\pi\in P(1^{n_1}\ldots m^{n_m})$ как произведение весов ее столбцов в двухстрочном представлении, где вес столбца $\binom{j}{k}$ равен w_j/w_k, если j и k

оба $\le t$ или оба $> t$, в противном случае вес равен z_j/z_k. Докажите, что сумма $w(\pi)$ по всем $\pi \in P(1^{n_1}\dots m^{n_m})$ равна

$$\sum_{k\ge 0} \frac{k!^2 (n_{\le t}-k)!\,(n_{>t}-k)!}{n_1!\dots n_m!} \left| \sum \binom{n_1}{p_1}\cdots\binom{n_m}{p_m}\left(\frac{w_1}{z_1}\right)^{p_1}\cdots\left(\frac{w_m}{z_m}\right)^{p_m}\right|^2,$$

где $n_{\le t}$ означает $n_1+\dots+n_t$, $n_{>t}$ означает $n_{t+1}+\dots+n_m$ и внутреннее суммирование выполняется по всем $(p_1,\dots,p_m)$, таким, что $p_{\le t}=p_{>t}=k$.

23. [*M23*] Цепочку ДНК можно рассматривать как слово четырехбуквенного алфавита. Предположим, мы скопировали цепочку ДНК и полностью разложили ее на однобуквенные составляющие, а затем случайным образом их вновь объединили. Докажите, что, если поместить полученную таким образом цепочку вслед за исходной, число позиций, в которых эти две цепочки будут отличаться, с большей вероятностью будет четным, чем нечетным. [*Указание.* Примените к этому случаю результат предыдущего упражнения.]

24. [*27*] Рассмотрим некоторое отношение R, которое может существовать между двумя неупорядоченными парами букв; если $\{w,x\}R\{y,z\}$, мы говорим, что $\{w,x\}$ *сохраняет* $\{y,z\}$, в противном случае $\{w,x\}$ *перемещает* $\{y,z\}$.

Операция *транспозиции* $\begin{smallmatrix} w & x \\ y & z \end{smallmatrix}$ применительно к R меняет $\begin{smallmatrix} w & x \\ y & z \end{smallmatrix}$ $\begin{smallmatrix} x & w \\ y & z \end{smallmatrix}$ или $\begin{smallmatrix} x & w \\ z & y \end{smallmatrix}$ в зависимости от того, сохраняет или перемещает пара $\{w,x\}$ пару $\{y,z\}$, полагая, что $w\ne x$ и $y\ne z$; если $w=x$ или $y=z$, то транспозиция всегда формирует $\begin{smallmatrix} x & w \\ z & y \end{smallmatrix}$.

Операция *сортировки* двухстрочного массива $\left(\begin{smallmatrix} x_1 & \dots & x_n \\ y_1 & \dots & y_n \end{smallmatrix}\right)$ применительно к R находит наибольшее x_j, такое, что $x_j > x_{j+1}$, и транспонирует (взаимно переставляет) столбцы j и $j+1$ до тех пор, пока не установится $x_1 \le \dots \le x_n$. (Мы не ставим условия, чтобы $y_1\dots y_n$ представляло собой перестановку $x_1\dots x_n$.)

a) Для данного $\left(\begin{smallmatrix} x_1 & \dots & x_n \\ y_1 & \dots & y_n \end{smallmatrix}\right)$ докажите, что для каждого $x\in\{x_1,\dots,x_n\}$ существует единственное $y\in\{y_1,\dots,y_n\}$, такое, что $\operatorname{sort}\left(\begin{smallmatrix} x_1 & \dots & x_n \\ y_1 & \dots & y_n \end{smallmatrix}\right) = \operatorname{sort}\left(\begin{smallmatrix} x & x'_2 & \dots & x'_n \\ y & y'_2 & \dots & y'_n \end{smallmatrix}\right)$ для некоторых $x'_2, y'_2, \dots, x'_n, y'_n$.

b) Обозначим через $\left(\begin{smallmatrix} w_1 & \dots & w_k \\ y_1 & \dots & y_k \end{smallmatrix}\right) ® \left(\begin{smallmatrix} x_1 & \dots & x_l \\ z_1 & \dots & z_l \end{smallmatrix}\right)$ результат сортировки $\left(\begin{smallmatrix} w_1 & \dots & w_k & x_1 & \dots & x_l \\ y_1 & \dots & y_k & z_1 & \dots & z_l \end{smallmatrix}\right)$ применительно к R. Например, если R всегда истинно, ® является просто сопоставлением, если R всегда ложно, ® представляет собой включающее произведение $\top$. Обобщите теорему A — докажите, что любая перестановка π мультимножества M имеет единственное представление вида

$$\pi = (x_{11}\dots x_{1n_1}\,y_1) ® \big((x_{21}\dots x_{2n_2}\,y_2) ® \dots ® (x_{t1}\dots x_{tn_t}\,y_t)\big),$$

удовлетворяющее (16), если переопределить цикл таким образом: в (11) вместо $(x_1\,x_2\,\dots\,x_n)$ подставить $(x_2\,\dots\,x_n\,x_1)$. Например, пусть $\{w,x\}R\{y,z\}$ означает, что w, x, y и z различны. Из этого, в свою очередь, следует, что разложение выражения (12) по аналогии с выражением (17) есть

$$(d\,d\,b\,c\,a) ® \big((c\,b\,b\,a) ® \big((c\,d\,b) ® \big((d\,b) ® (d)\big)\big)\big).$$

(Операция ® отнюдь не всегда следует закону ассоциативности; скобки в обобщенном разложении следует раскрывать справа налево.)

*5.1.3. Серии

В главе 3 была проанализирована длина неубывающих серий в перестановке и показано, что этот параметр позволяет проверить случайность последовательности. Если поместить вертикальные черточки до и после перестановки $a_1\,a_2\dots a_n$, а также между a_j и a_{j+1}, когда $a_j > a_{j+1}$, то *сериями* будут называться серии, ограниченные парами черточек. Например, в перестановке

имеется четыре серии. В разделе 3.3.2G были найдены среднее число серий длины k в случайной перестановке множества $\{1, 2, \ldots, n\}$ и ковариация числа серий длины j и длины k. Серии важны при изучении алгоритмов сортировки, так как они представляют упорядоченные серии данных. Поэтому-то мы теперь вновь вернемся к вопросу о сериях.

Обозначим через

$$\left\langle {n \atop k} \right\rangle \tag{1}$$

число перестановок множества $\{1, 2, \ldots, n\}$, имеющих ровно k нисходящих серий $a_j > a_{j+1}$ и $k+1$ восходящих серий. Такие числа $\left\langle {n \atop k} \right\rangle$ возникают и в других контекстах; их обычно называют *числами Эйлера*, потому что Эйлер проанализировал их в своей знаменитой книге *Institutiones Calculi Differentialis* (St. Petersburg: 1755, 485–487) после того, как впервые использовал несколькими годами ранее [*Comment. Acad. Sci. Imp. Petrop.* **8** (1736), 147–158, §13]; их следует отличать от *чисел Эйлера* E_n, о которых идет речь в упр. 5.1.4–23. Угловые скобки в $\left\langle {n \atop k} \right\rangle$ напоминают символ "$>$", который присутствует в определении убывания. Конечно, $\left\langle {n \atop k} \right\rangle$ также равно числу перестановок, в которых имеется k возрастаний $a_j < a_{j+1}$.

Из любой данной перестановки множества $\{1, \ldots, n-1\}$ можно образовать n новых перестановок, вставляя элемент n во все возможные места. Если в исходной перестановке содержалось k нисходящих серий, то ровно $k+1$ новых перестановок будут иметь k нисходящих серий; в остальных $n-1-k$ перестановках будет по $k+1$ серий, поскольку всякий раз, когда n вставляется не в конец существующей серии, число нисходящих серий увеличивается на единицу. Например, из перестановки 3 1 2 4 5 можно получить шесть перестановок

$$\begin{array}{lll} 6\,3\,1\,2\,4\,5, & 3\,6\,1\,2\,4\,5, & 3\,1\,6\,2\,4\,5, \\ 3\,1\,2\,6\,4\,5, & 3\,1\,2\,4\,6\,5, & 3\,1\,2\,4\,5\,6. \end{array}$$

Все эти перестановки, кроме второй и последней, содержат по две нисходящие серии вместо одной исходной. Отсюда имеем рекуррентное соотношение

$$\left\langle {n \atop k} \right\rangle = (k+1) \left\langle {n-1 \atop k} \right\rangle + (n-k) \left\langle {n-1 \atop k-1} \right\rangle, \tag{2}$$

где целое $n > 0$ и k также целое.

Условимся, что

$$\left\langle {0 \atop k} \right\rangle = \delta_{k0}\,, \tag{3}$$

т. е. будем считать, что в нуль-перестановке (пустой перестановке) не содержатся нисходящие серии. Читатель, возможно, найдет небезынтересным сравнение (2) с рекуррентным соотношением для чисел Стирлинга [формулы 1.2.6–(46)]. В табл. 1 приведены числа Эйлера для малых n.

В табл. 1 можно заметить некоторые закономерности. По определению имеем

$$\left\langle {n \atop 0} \right\rangle + \left\langle {n \atop 1} \right\rangle + \cdots + \left\langle {n \atop n} \right\rangle = n!\,; \tag{4}$$

$$\left\langle {n \atop 0} \right\rangle = 1\,; \tag{5}$$

Таблица 1

ЧИСЛА ЭЙЛЕРА

n	$\left\langle {n \atop 0} \right\rangle$	$\left\langle {n \atop 1} \right\rangle$	$\left\langle {n \atop 2} \right\rangle$	$\left\langle {n \atop 3} \right\rangle$	$\left\langle {n \atop 4} \right\rangle$	$\left\langle {n \atop 5} \right\rangle$	$\left\langle {n \atop 6} \right\rangle$	$\left\langle {n \atop 7} \right\rangle$	$\left\langle {n \atop 8} \right\rangle$
0	1	0	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0
2	1	1	0	0	0	0	0	0	0
3	1	4	1	0	0	0	0	0	0
4	1	11	11	1	0	0	0	0	0
5	1	26	66	26	1	0	0	0	0
6	1	57	302	302	57	1	0	0	0
7	1	120	1191	2416	1191	120	1	0	0
8	1	247	4293	15619	15619	4293	247	1	0
9	1	502	14608	88234	156190	88234	14608	502	1

$$\left\langle {n \atop n-1} \right\rangle = 1\,, \qquad \left\langle {n \atop n} \right\rangle = 0\,, \qquad \text{где } n \geq 1. \tag{6}$$

Соотношение (6) следует из (5) вследствие свойства симметрии

$$\left\langle {n \atop k} \right\rangle = \left\langle {n \atop n-1-k} \right\rangle, \qquad \text{где } n \geq 1, \tag{7}$$

которое вытекает из того факта, что каждая непустая перестановка $a_1\, a_2 \ldots a_n$, содержащая k нисходящих серий, имеет и $n-1-k$ восходящих серий.

Другое важное свойство чисел Эйлера выражается формулой

$$\sum_k \left\langle {n \atop k} \right\rangle \binom{m+k}{n} = m^n\,, \qquad n \geq 0\,, \tag{8}$$

которая была впервые выведена китайским математиком Ли Шан-Ланом и опубликована в 1867 году. [См. J.-C. Martzloff, *A History of Chinese Mathematics* (Berlin: Springer, 1997, 346–348); особый случай, если $n \leq 5$, был независимо рассмотрен японским математиком Йошисуке Мацунага (Matsunaga Yohisuke), который умер в 1744 году.] Тождество Ли Шан-Лана следует из свойств операции сортировки. Рассмотрим m^n последовательностей $a_1\, a_2 \ldots a_n$, где $1 \leq a_i \leq m$. Любую такую последовательность можно устойчиво рассортировать в порядке неубывания и получить:

$$a_{i_1} \leq a_{i_2} \leq \cdots \leq a_{i_n}\,, \tag{9}$$

где $i_1\, i_2 \ldots i_n$ — однозначно определенная перестановка множества $\{1, 2, \ldots, n\}$, такая, что $i_j < i_{j+1}$, если $a_{i_j} = a_{i_{j+1}}$; другими словами, из $i_j > i_{j+1}$ следует $a_{i_j} < a_{i_{j+1}}$. Покажем, что если в перестановке $i_1\, i_2 \ldots i_n$ содержится k серий, то число соответствующих ей последовательностей $a_1\, a_2 \ldots a_n$ равно $\binom{m+n-k}{n}$; тем самым будет доказана формула (8), если заменить k значением $n-k$ и воспользоваться (7), поскольку $\left\langle {n \atop k} \right\rangle$ перестановок имеют $n-k$ серий.

Пусть, например, $n = 9$, $i_1\, i_2 \ldots i_n = 3\,5\,7\,1\,6\,8\,9\,4\,2$ и требуется подсчитать число последовательностей $a_1\, a_2 \ldots a_n$, таких, что

$$1 \leq a_3 \leq a_5 \leq a_7 < a_1 \leq a_6 \leq a_8 \leq a_9 < a_4 < a_2 \leq m\,. \tag{10}$$

Оно равно числу последовательностей $b_1\, b_2 \ldots b_9$, таких, что

$$1 \le b_1 < b_2 < b_3 < b_4 < b_5 < b_6 < b_7 < b_8 < b_9 \le m+5\,,$$

поскольку можно положить $b_1 = a_3$, $b_2 = a_5 + 1$, $b_3 = a_7 + 2$, $b_4 = a_1 + 2$, $b_5 = a_6 + 3$ и т. д. Число способов, которыми можно выбрать элементы b, равно просто-напросто числу способов выбора 9 предметов из $m+5$, т. е. $\binom{m+5}{9}$; аналогичное доказательство годится для произвольных n и k и любой перестановки $i_1\, i_2 \ldots i_n$ с k сериями.

Так как в обеих частях равенства (8) стоят полиномы от m, вместо m можно подставить любое действительное число, получив интересное выражение степеней через последовательные биномиальные коэффициенты:

$$x^n = \left\langle {n \atop 0} \right\rangle \binom{x}{n} + \left\langle {n \atop 1} \right\rangle \binom{x+1}{n} + \cdots + \left\langle {n \atop n-1} \right\rangle \binom{x+n-1}{n}, \quad n \ge 1. \tag{11}$$

Например,

$$x^3 = \binom{x}{3} + 4\binom{x+1}{3} + \binom{x+2}{3}.$$

В основном благодаря именно этому свойству числа Эйлера весьма широко применяются в дискретной математике. Положив в (11) $x = 1$, докажем еще раз, что $\left\langle {n \atop n-1} \right\rangle = 1$. поскольку биномиальные коэффициенты обращаются в 0 во всех слагаемых, кроме последнего. Положив $x = 2$, получим

$$\left\langle {n \atop n-2} \right\rangle = \left\langle {n \atop 1} \right\rangle = 2^n - n - 1\,, \qquad n \ge 1. \tag{12}$$

Подставив $x = 3,\ 4,\ \ldots$, убедимся, что все числа $\left\langle {n \atop k} \right\rangle$ полностью определяются соотношением (11), и придем к формуле, впервые найденной Эйлером:

$$\begin{aligned}\left\langle {n \atop k} \right\rangle &= (k+1)^n - k^n\binom{n+1}{1} + (k-1)^n\binom{n+1}{2} - \cdots + (-1)^k 1^n \binom{n+1}{k} \\ &= \sum_{j=0}^{k} (-1)^j \binom{n+1}{j} (k+1-j)^n, \qquad n \ge 0,\ k \ge 0. \end{aligned} \tag{13}$$

Рассмотрим теперь производящую функцию для серий. Если положить

$$g_n(z) = \sum_k \left\langle {n \atop k-1} \right\rangle \frac{z^k}{n!}\,, \tag{14}$$

то коэффициент при z^k будет равен вероятности того, что случайная перестановка множества $\{1, 2, \ldots, n\}$ содержит ровно k серий. Поскольку k серий в перестановке столь же вероятны, как и $n+1-k$, среднее число серий должно равняться $\frac{1}{2}(n+1)$ и, следовательно, $g_n'(1) = \frac{1}{2}(n+1)$. В упр. 2, (b) показано, что имеет место простая формула для *всех* производных функции $g_n(z)$ в точке $z = 1$:

$$g_n^{(m)}(1) = \left\{ {n+1 \atop n+1-m} \right\} \Big/ \binom{n}{m}, \qquad n \ge m. \tag{15}$$

Так, в частности, дисперсия $g_n''(1) + g_n'(1) - g_n'(1)^2$ равна $(n+1)/12$ при $n \ge 2$, что указывает на довольно устойчивое распределение около среднего значения. (Эта же величина была найдена в упр. 3.3.2–(18); в нем она называлась $\operatorname{covar}(R_1', R_1')$.)

Функция $g_n(z)$ — полином, поэтому с помощью формулы (15) и формулы Тейлора ее можно представить в виде

$$g_n(z) = \frac{1}{n!}\sum_{k=0}^{n}(z-1)^{n-k}k!\left\{{n+1 \atop k+1}\right\} = \frac{1}{n!}\sum_{k=0}^{n}z^{k+1}(1-z)^{n-k}k!\left\{{n+1 \atop k+1}\right\}. \tag{16}$$

Второе равенство следует из первого, поскольку вследствие условия симметрии (7)

$$g_n(z) = z^{n+1}g_n(1/z), \qquad n \geq 1. \tag{17}$$

Из рекуррентного соотношения для чисел Стирлинга

$$\left\{{n+1 \atop k+1}\right\} = (k+1)\left\{{n \atop k+1}\right\} + \left\{{n \atop k}\right\}$$

при $n \geq 1$ получаются два более простых представления:

$$g_n(z) = \frac{1}{n!}\sum_{k=0}^{n}z(z-1)^{n-k}k!\left\{{n \atop k}\right\} = \frac{1}{n!}\sum_{k=0}^{n}z^{k}(1-z)^{n-k}k!\left\{{n \atop k}\right\}. \tag{18}$$

Производящая функция от двух переменных*

$$g(z,x) = \sum_{n\geq 0}\frac{g_n(z)x^n}{z} = \sum_{k,n\geq 0}\left\langle{n \atop k}\right\rangle\frac{z^k x^n}{n!} \tag{19}$$

равна, следовательно,

$$\sum_{k,n\geq 0}\frac{((z-1)x)^n}{(z-1)^k}\left\{{n \atop k}\right\}\frac{k!}{n!} = \sum_{k\geq 0}\left(\frac{e^{(z-1)x}-1}{z-1}\right)^k = \frac{(1-z)}{e^{(z-1)x}-z}. \tag{20}$$

Это еще одно соотношение, проанализированное Эйлером.

Другие свойства чисел Эйлера можно найти в обзорной статье L. Carlitz, *Math. Magazine* **33** (1959), 247–260. (См. также работы J. Riordan, *Introduction to Combinatorial Analysis* (New York: Wiley, 1958, 38–39), 214–219, 234–237**; D. Foata, M. P. Schützenberger, *Lecture Notes in Math.* **138** (Berlin: Springer, 1970).)

Рассмотрим теперь длину серий; какова в среднем длина серии? В разделе 3.3.2 уже было проанализировано математическое ожидание числа серий данной длины; средняя длина серии равна примерно 2. Это согласуется с тем фактом, что в случайной перестановке длины n содержится около $\frac{1}{2}(n+1)$ серий. Применительно к алгоритмам сортировки полезна несколько отличная точка зрения; рассмотрим длину k-й слева серии перестановки при $k = 1, 2, \ldots$.

Какова, например, длина первой (крайней слева) серии случайной перестановки $a_1\, a_2 \ldots a_n$? Ее длина всегда ≥ 1; она ≥ 2 ровно в половине случаев (а именно, если $a_1 < a_2$). Ее длина ≥ 3 ровно для $1/6$ случаев (если $a_1 < a_2 < a_3$). Вообще, ее длина $\geq m$ с вероятностью $q_m = 1/m!$ для $1 \leq m \leq n$. Следовательно, вероятность того, что длина этой серии равна в точности m, есть

$$\begin{aligned} p_m &= q_m - q_{m+1} = 1/m! - 1/(m+1)!, \qquad \text{где } 1 \leq m < n; \\ p_n &= 1/n!. \end{aligned} \tag{21}$$

---

* В оригинале — "super generating function". — *Прим. перев.*

** Имеется русский перевод: Риордан Дж. Введение в комбинаторный анализ. — М.: Изд-во иностр. лит., 1963. — *Прим. перев.*

Средняя длина первой серии равна, таким образом,

$$\begin{aligned} p_1 + 2p_2 + \cdots + np_n &= (q_1 - q_2) + 2(q_2 - q_3) + \cdots + (n-1)(q_{n-1} - q_n) + nq_n \\ &= q_1 + q_2 + \cdots + q_n = \frac{1}{1!} + \frac{1}{2!} + \cdots + \frac{1}{n!}. \end{aligned} \tag{22}$$

Предел при $n \to \infty$ равен $e - 1 = 1.71828\ldots$, а для конечных n это значение равно $e - 1 - \delta_n$, где δ_n довольно мало (для больших n. — *Прим. ред.*):

$$\delta_n = \frac{1}{(n+1)!}\left(1 + \frac{1}{n+2} + \frac{1}{(n+2)(n+3)} + \cdots\right) \le \frac{e-1}{(n+1)!}.$$

Поэтому для практических целей удобно рассмотреть серии *случайной бесконечной* последовательности различных чисел

$$a_1, a_2, a_3, \ldots;$$

под "случайностью" последовательности здесь подразумевается то, что все $n!$ возможных взаимных расположений первых n элементов последовательности равновероятны (при любом n. — *Прим. ред.*). Так что средняя длина первой серии случайной бесконечной последовательности в точности равна $e - 1$.

Несколько усовершенствовав этот метод, можно установить среднюю длину k-й серии случайной последовательности. Пусть q_{km} — вероятность того, что общая длина первых k серий $\ge m$; тогда q_{km} равно величине $1/m!$, умноженной на число перестановок множества $\{1, 2, \ldots, m\}$, которые содержат не более k серий:

$$q_{km} = \left(\left\langle {m \atop 0} \right\rangle + \cdots + \left\langle {m \atop k-1} \right\rangle\right) \Big/ m!. \tag{23}$$

Вероятность того, что общая длина первых k серий равна m, есть $q_{km} - q_{k(m+1)}$. Следовательно, обозначив через L_k среднюю длину k-й серии, находим, что

$$\begin{aligned} L_1 + \cdots + L_k &= \text{средняя общая длина первых } k \text{ серий} \\ &= (q_{k1} - q_{k2}) + 2(q_{k2} - q_{k3}) + 3(q_{k3} - q_{k4}) + \cdots \\ &= q_{k1} + q_{k2} + q_{k3} + \cdots. \end{aligned}$$

Вычитая $L_1 + \cdots + L_{k-1}$ и используя значения q_{km}, из (23) получаем нужную нам формулу:

$$L_k = \frac{1}{1!}\left\langle {1 \atop k-1} \right\rangle + \frac{1}{2!}\left\langle {2 \atop k-1} \right\rangle + \frac{1}{3!}\left\langle {3 \atop k-1} \right\rangle + \cdots = \sum_{m \ge 1} \left\langle {m \atop k-1} \right\rangle \frac{1}{m!}. \tag{24}$$

Поскольку $\left\langle {0 \atop k-1} \right\rangle = 0$, кроме случая $k = 1$, значение L_k оказывается равным коэффициенту при z^{k-1} в производящей функции $g(z, 1) - 1$ (см. (19)); таким образом, имеем

$$L(z) = \sum_{k \ge 0} L_k z^k = \frac{z(1-z)}{e^{z-1} - z} - z. \tag{25}$$

Из формулы Эйлера (13) получим представление L_k в виде полинома от e:

$$\begin{aligned}
L_k &= \sum_{m\ge 0}\sum_{j=0}^{k}(-1)^{k-j}\binom{m+1}{k-j}\frac{j^m}{m!}\\
&= \sum_{j=0}^{k}(-1)^{k-j}\sum_{m\ge 0}\binom{m}{k-j}\frac{j^m}{m!}+\sum_{j=0}^{k}(-1)^{k-j}\sum_{m\ge 0}\binom{m}{k-j-1}\frac{j^m}{m!}\\
&= \sum_{j=0}^{k}\frac{(-1)^{k-j}j^{k-j}}{(k-j)!}\sum_{n\ge 0}\frac{j^n}{n!}+\sum_{j=0}^{k}\frac{(-1)^{k-j}j^{k-j-1}}{(k-j-1)!}\sum_{n\ge 0}\frac{j^n}{n!}\\
&= k\sum_{j=0}^{k}\frac{(-1)^{k-j}j^{k-j-1}}{(k-j)!}e^j. \qquad (26)
\end{aligned}$$

Эту формулу для L_k впервые получила Б. Дж. Гэсснер (B. J. Gassner) [см. *CACM* **10** (1967), 89–93]. Имеем, в частности,

$$\begin{aligned}
L_1 &= e-1 &&\approx 1.71828\ldots;\\
L_2 &= e^2-2e &&\approx 1.95249\ldots;\\
L_3 &= e^3-3e^2+\tfrac{3}{2}e &&\approx 1.99579\ldots.
\end{aligned}$$

Итак, следует ожидать, что вторая серия будет длиннее первой, а третья серия будет в среднем еще длиннее! На первый взгляд, это может показаться странным, но после минутного размышления становится ясно, что, поскольку первый элемент второй серии будет, скорее всего, малым числом (именно это служит причиной окончания первой серии), у второй серии больше шансов оказаться длиннее, чем у первой. Тенденция такова, что первый элемент третьей серии будет даже меньше, чем первый элемент второй серии. Числа L_k важны в теории сортировки посредством выбора с замещением (раздел 5.4.1), поэтому интересно подробно проанализировать их значения. В табл. 2 приведены первые 18 значений L_k с точностью до 15 десятичных знаков. Рассуждения, приведенные в предыдущем абзаце, могут поначалу вызвать подозрение, что $L_{k+1} > L_k$, на самом же деле значения колеблются, то возрастая, то убывая. Заметим, что L_k быстро приближаются к предельному значению 2. Весьма примечательно то, что эти нормированные полиномы от трансцендентного числа e так быстро сходятся к рациональному числу 2! Полиномы (26) представляют некоторый интерес и с точки зрения численного анализа, поскольку являются прекрасным примером потери значащих цифр при вычитании почти равных чисел; используя арифметику с плавающей точкой и представление с точностью до 19 значащих разрядов, Гэсснер пришла к неверному заключению о том, что $L_{12} > 2$, а Джон У. Ренч (мл.) (John W. Wrench, Jr.) отметил, что выполнение арифметических операций с точностью до 42 значащих разрядов дает значения L_{28} с точностью только до 29 значащих цифр.

Асимптотическое поведение L_k можно определить, исходя из простых положений теории функций комплексного переменного. Знаменатель в (25) обращается в нуль лишь при $e^{z-1} = z$, т. е. когда

$$e^{x-1}\cos y = x \qquad \text{и} \qquad e^{x-1}\sin y = y, \qquad (27)$$

Таблица 2

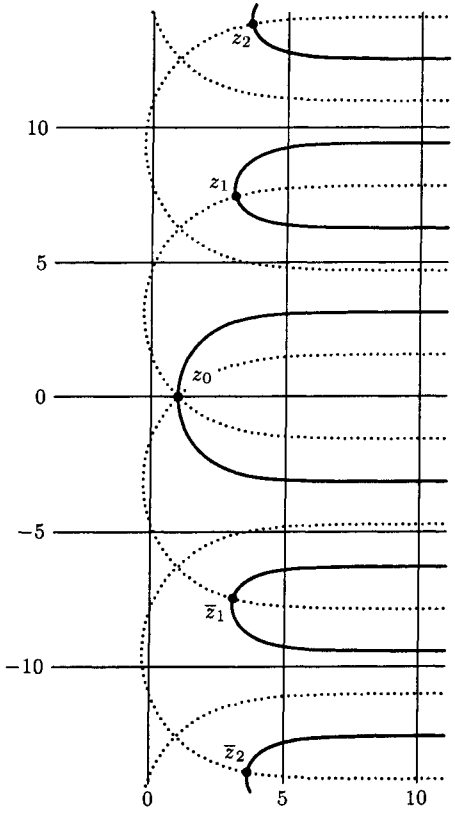
СРЕДНЯЯ ДЛИНА k-Й СЕРИИ

k	L_k	k	L_k
1	1.71828 18284 59045+	10	2.00000 00012 05997+
2	1.95249 24420 12560−	11	2.00000 00001 93672+
3	1.99579 13690 84285−	12	1.99999 99999 99909+
4	2.00003 88504 76806−	13	1.99999 99999 97022−
5	2.00005 75785 89716+	14	1.99999 99999 99719+
6	2.00000 50727 55710−	15	2.00000 00000 00019+
7	1.99999 96401 44022+	16	2.00000 00000 00006+
8	1.99999 98889 04744+	17	2.00000 00000 00000+
9	1.99999 99948 43434−	18	2.00000 00000 00000−

полагая $z = x + iy$. На рис. 3, на котором нанесены оба графика этих уравнений, видно, что они пересекаются в точках $z = z_0, z_1, \bar{z}_1, z_2, \bar{z}_2, \ldots$, где $z_0 = 1$,

$$z_1 = (3.08884\ 30156\ 13044-) + (7.46148\ 92856\ 54255-)\, i, \tag{28}$$

и при больших k мнимая часть $\Im(z_{k+1})$ равна приблизительно $\Im(z_k) + 2\pi$.



$e^{x-1}\sin y = y$ ———

$e^{x-1}\cos y = x$ ·····

Рис. 3. Корни уравнения $e^{z-1} = z$.

Так как

$$\lim_{z\to z_k}\left(\frac{1-z}{e^{z-1}-z}\right)(z-z_k)=-1 \qquad \text{при } k>0$$

и этот предел равен -2 при $k=0$, функция

$$R_m(z)=L(z)+\frac{2}{z-z_0}+\frac{z_1}{z-z_1}+\frac{\bar z_1}{z-\bar z_1}+\frac{z_2}{z-z_2}+\frac{\bar z_2}{z-\bar z_2}+\cdots+\frac{z_m}{z-z_m}+\frac{\bar z_m}{z-\bar z_m}$$

не имеет особых точек на комплексной плоскости при $|z|<|z_{m+1}|$. Значит, $R_m(z)$ можно разложить в степенной ряд $\sum_k \rho_k z^k$, который сходится абсолютно при $|z|<|z_{m+1}|$; отсюда следует, что $\rho_k M^k\to 0$ при $k\to\infty$, где $M=|z_{m+1}|-\epsilon$. Коэффициентами для $L(z)$ служат коэффициенты разложения функции

$$\frac{2}{1-z}+\frac{1}{1-z/z_1}+\frac{1}{1-z/\bar z_1}+\cdots+\frac{1}{1-z/z_m}+\frac{1}{1-z/\bar z_m}+R_m(z),$$

а именно

$$L_n=2+2r_1^{-n}\cos n\theta_1+2r_2^{-n}\cos n\theta_2+\cdots+2r_m^{-n}\cos n\theta_m+O(r_{m+1}^{-n}), \tag{29}$$

если положить

$$z_k=r_k e^{i\theta_k}. \tag{30}$$

Отсюда можно проследить асимптотическое поведение L_n. Имеем

$$\begin{aligned}
r_1 &= 8.07556\,64528\,89526-, & \theta_1 &= 1.17830\,39784\,74668+;\\
r_2 &= 14.35456\,68997\,62106-, & \theta_2 &= 1.31268\,53883\,87636+;\\
r_3 &= 20.62073\,15381\,80628-, & \theta_3 &= 1.37427\,90757\,91688-;\\
r_4 &= 26.88795\,29424\,54546-, & \theta_4 &= 1.41049\,72786\,51865-;
\end{aligned} \tag{31}$$

таким образом, главный вклад в L_n-2 дают r_1 и θ_1, а ряд (29) сходится очень быстро. Дальнейший анализ [W. W. Hooker, *CACM* **12** (1969), 411–413] показывает, что $R_m(z)\to -z$ при $m\to\infty$; следовательно, ряд $2\sum_{k\ge 0} r_k^{-n}\cos n\theta_k$ действительно *сходится* к L_n при $n>1$.

Можно провести более тщательный анализ и полностью определить распределение вероятностей для длины k-й серии и для общей длины первых k серий (см. упр. 9–11). Оказывается, сумма $L_1+\cdots+L_k$ асимптотически приближается к $2k-\frac{1}{3}+O(8^{-k})$.

В заключение этого раздела рассмотрим свойства серий в случае, когда в перестановке допускаются одинаковые элементы. Бесчисленные пасьянсы, которым посвящал свой досуг знаменитый американский астроном 19 в. Саймон Ньюкомб (Simon Newcomb), имеют непосредственное отношение к интересующему нас вопросу. Он брал колоду карт и складывал их в одну стопку до тех пор, пока они шли в порядке неубывания по старшинству; как только следующая карта оказывалась младше предыдущей, он начинал новую стопку. Он хотел найти такую вероятность, при которой вся колода окажется разложенной на заданное количество стопок. Задача Саймона Ньюкомба состоит, следовательно, в нахождении распределения вероятностей для серий случайной перестановки мультимножества. В общем случае ответ довольно сложен (см. упр. 12), хотя мы уже видели, как решать задачу в частном случае, когда все карты различны по старшинству. Мы удовлетворимся здесь выводом формулы для *среднего* числа стопок в этом пасьянсе.

Пусть имеется m различных типов карт и каждая встречается ровно p раз. Например, в обычной колоде для бриджа $m = 13$, а $p = 4$, если пренебрегать различием масти. Замечательную симметрию обнаружил в этом случае П. А. Мак-Магон [см. *Combinatory Analysis* **1** (Cambridge, 1915, 212–213)]: число перестановок с $k+1$ сериями равно числу перестановок с $mp-p-k+1$ сериями. Это соотношение легко проверить при $p = 1$ (формула (7)), однако при $p > 1$ оно кажется довольно неожиданным.

Можно доказать свойство симметрии, установив взаимно однозначное соответствие между перестановками, такое, что каждой перестановке с $k+1$ сериями соответствует перестановка с $mp-p-k+1$ сериями. Мы настойчиво рекомендуем читателю постараться самостоятельно найти такое соответствие, прежде чем двигаться дальше.

Какое-нибудь очень простое соответствие на ум не приходит; доказательство Мак-Магона основано на производящих функциях, а не на комбинаторном построении. Однако установленное Фоатой соответствие (теорема 5.1.2B) позволяет упростить задачу, так как в нем утверждается существование взаимно однозначного соответствия между перестановками с $k+1$ сериями и перестановками, в двухстрочном представлении которых содержится ровно k столбцов $\frac{y}{x}$, таких, что $x < y$.

Пусть дано мультимножество $\{p\cdot 1, p\cdot 2, \ldots, p\cdot m\}$. Рассмотрим перестановку в двухстрочном представлении

$$\begin{pmatrix} 1 & \ldots & 1 & 2 & \ldots & 2 & \ldots & m & \ldots & m \\ x_{11} & \ldots & x_{1p} & x_{21} & \ldots & x_{2p} & \ldots & x_{m1} & \ldots & x_{mp} \end{pmatrix}. \tag{32}$$

Сопоставим с ней другую перестановку того же мультимножества:

$$\begin{pmatrix} 1 & \ldots & 1 & 2 & \ldots & 2 & \ldots & m & \ldots & m \\ x'_{11} & \ldots & x'_{1p} & x'_{m1} & \ldots & x'_{mp} & \ldots & x'_{21} & \ldots & x'_{2p} \end{pmatrix}, \tag{33}$$

где $x' = m+1-x$. Если перестановка (32) содержит k столбцов вида $\frac{y}{x}$, таких, что $x < y$, то (33) содержит $(m-1)p-k$ таких столбцов, потому что необходимо рассмотреть лишь случай $y > 1$, а неравенство $x < y$ эквивалентно $x' \geq m+2-y$. Поскольку (32) соответствует перестановке с $k+1$ сериями, а (33) — перестановке с $mp-p-k+1$ сериями и преобразование, переводящее (32) в (33), обратимо (оно же переводит (33) в (32)), то тем самым доказано условие симметрии Мак-Магона. Пример этого построения содержится в упр. 14.

В силу свойства симметрии среднее число серий в случайной перестановке должно равняться $\frac{1}{2}\big((k+1)+(mp-p-k+1)\big) = 1+\frac{1}{2}p(m-1)$. Например, для стандартной колоды среднее число стопок в пасьянсе Ньюкомба равно 25 (так что раскладывание этого пасьянса вряд ли покажется столь уж увлекательным занятием).

На самом деле после несложных рассуждений можно определить среднее число серий в общем случае для *любого* данного мультимножества $\{n_1\cdot x_1, n_2\cdot x_2, \ldots, n_m\cdot x_m\}$, где все x_k различны. Пусть $n = n_1+n_2+\cdots+n_m$ и все перестановки $a_1\,a_2\ldots a_n$ этого мультимножества выписаны в явном виде. Посмотрим, как часто a_i оказывается больше a_{i+1} при каждом фиксированном значении i, $1 \leq i < n$. Число случаев, когда $a_i > a_{i+1}$, равно в точности половине числа случаев, когда $a_i \neq a_{i+1}$, и нетрудно видеть, что $a_i = a_{i+1} = x_j$ ровно в $Nn_j(n_j-1)/n(n-1)$ случаях, где N — общее число перестановок. Следовательно, $a_i = a_{i+1}$ ровно в

$$\frac{N}{n(n-1)}\big(n_1(n_1-1)+\cdots+n_m(n_m-1)\big)=\frac{N}{n(n-1)}(n_1^2+\cdots+n_m^2-n)$$

случаях, а $a_i > a_{i+1}$ в

$$\frac{N}{2n(n-1)}\big(n^2-(n_1^2+\cdots+n_m^2)\big)$$

случаях. Суммируя по i и прибавляя N, потому что в каждой перестановке одна серия заканчивается элементом a_n, получим общее число серий во всех N перестановках:

$$N\Big(\frac{n}{2}-\frac{1}{2n}(n_1^2+\cdots+n_m^2)+1\Big). \tag{34}$$

Выполнив деление на N, получим искомое среднее число серий.

Поскольку серии играют весьма важную роль в изучении "порядковых статистик", имеется весьма обширный список работ, посвященных этой теме, включая некоторые типы серий, не рассмотренные здесь. Дополнительную информацию можно найти в книге F. N. David, D. E. Barton, Combinatorial Chance (London: Griffin 1962, гл. 10) и в обзорной статье C. L. Mallows, *Annals of Math. Statistics* **36** (1965), 236–260.

УПРАЖНЕНИЯ

1. [*M26*] Выведите формулу Эйлера (13).

▶ **2.** [*M22*] (a) Попытайтесь развить идею, использованную в тексте при доказательстве тождества (8). Рассмотрите последовательности $a_1\,a_2\ldots a_n$, содержащие ровно q различных элементов, и докажите, что

$$\sum_k \left\langle {n \atop k} \right\rangle \binom{k}{n-q} = \left\{ {n \atop q} \right\} q!.$$

(b) Используя это тождество, докажите, что

$$\sum_k \left\langle {n \atop k} \right\rangle \binom{k+1}{m} = \left\{ {n+1 \atop n+1-m} \right\} (n-m)! \qquad \text{при } n \ge m.$$

3. [*HM25*] Вычислите сумму $\sum_k \langle {n \atop k} \rangle (-1)^k$.

4. [*M21*] Чему равна сумма $\sum_k (-1)^k \left\{ {n \atop k} \right\} k! \binom{n-k}{m}$?

5. [*M20*] Найдите значение $\langle {p \atop k} \rangle \bmod p$, если p — простое число.

▶ **6.** [*M21*] Мистер Далл заметил, что из формул (4) и (13) можно получить

$$n! = \sum_{k\ge 0} \left\langle {n \atop k} \right\rangle = \sum_{k\ge 0}\sum_{j\ge 0} (-1)^{k-j} \binom{n+1}{k-j} (j+1)^n.$$

Он обнаружил, что $\sum_{k\ge 0}(-1)^{k-j}\binom{n+1}{k-j} = 0$ при всех $j \ge 0$, выполнив суммирование сначала по k, а затем по $j \ge 0$; отсюда $n! = 0$ при любом $n \ge 0$. Не допустил ли он какой-нибудь ошибки?

7. [*HM40*] Является ли распределение вероятностей для серий, задаваемое формулой (14), асимптотически нормальным? (Ср. с упр. 1.2.10–13.)

8. [*M24*] (П. А. Мак-Магон.) Покажите, что вероятность того, что длина первой серии достаточно длинной перестановки есть l_1, длина второй есть l_2, ..., а длина k-й серии $\ge l_k$ равна

$$\det\begin{pmatrix} 1/l_1! & 1/(l_1+l_2)! & 1/(l_1+l_2+l_3)! & \dots & 1/(l_1+l_2+l_3+\cdots+l_k)! \\ 1 & 1/l_2! & 1/(l_2+l_3)! & \dots & 1/(l_2+l_3+\cdots+l_k)! \\ 0 & 1 & 1/l_3! & \dots & 1/(l_3+\cdots+l_k)! \\ \vdots & & & & \vdots \\ 0 & 0 & \dots & 1 & 1/l_k! \end{pmatrix}.$$

9. [*M30*] Пусть $h_k(z) = \sum p_{km} z^m$, где p_{km} — вероятность того, что общая длина первых k серий случайной последовательности (бесконечной) равна m. Найдите "простые" выражения для $h_1(z)$, $h_2(z)$ и для производящей функции $h(z,x) = \sum_k h_k(z) x^k$ двух переменных.

10. [*HM30*] Определите асимптотическое поведение математического ожидания и дисперсии распределения $h_k(z)$ из предыдущего упражнения при больших k.

11. [*M40*] Пусть $H_k(z) = \sum P_{km} z^m$, где P_{km} — вероятность того, что длина k-й серии в случайной (бесконечной) последовательности равна m. Выразите $H_1(z)$, $H_2(z)$ и производящую функцию $H(z,x) = \sum_k H_k(z) x^k$ от двух переменных через известные функции.

12. [*M33*] (П. А. Мак-Магон.) Обобщите формулу (13) на случай перестановок мультимножества, доказав, что число перестановок мультимножества $\{n_1 \cdot 1, n_2 \cdot 2, \dots, n_m \cdot m\}$, имеющих ровно k серий, равно

$$\sum_{j=0}^{k} (-1)^j \binom{n+1}{j} \binom{n_1 - 1 + k - j}{n_1} \binom{n_2 - 1 + k - j}{n_2} \dots \binom{n_m - 1 + k - j}{n_m},$$

где $n = n_1 + n_2 + \cdots + n_m$.

13. [*05*] Каким будет среднее число стопок в пасьянсе Ньюкомба, если пользоваться обычной колодой для бриджа (из 52 карт), игнорируя старшинство карт, но считая, что трефы < бубны < черви < пики?

14. [*M18*] Перестановка 3 1 1 1 2 3 1 4 2 3 3 4 2 2 4 4 содержит 5 серий; найдите соответствующую перестановку с 9-ю сериями с помощью построения для условия симметрии Мак-Магона, приведенного в тексте раздела.

▸ **15.** [*M21*] (*Перемежающиеся серии.*) В классической литературе 19 в. по комбинаторному анализу рассматриваемый нами вопрос о сериях в перестановках не изучался, но некоторые авторы изучали попеременно восходящие и нисходящие серии. Так, считалось, что перестановка 5 3 2 4 7 6 1 8 содержит 4 серии: 5 3 2, 2 4 7, 7 6 1 и 1 8. (Первая серия будет восходящей или нисходящей в зависимости от того, $a_1 < a_2$ или $a_1 > a_2$; таким образом, перестановки $a_1 a_2 \dots a_n$, $a_n \dots a_2 a_1$ и $(n+1-a_1)(n+1-a_2)\dots(n+1-a_n)$ содержат одинаковое число перемежающихся серий.) Максимальное число серий этого типа в перестановке n элементов равно $n-1$.

Найдите среднее число перемежающихся серий в случайной перестановке множества $\{1, 2, \dots, n\}$. [*Указание.* Разберите вывод формулы (34).]

16. [*M30*] Продолжим предыдущее упражнение. Пусть $\left|{n \atop k}\right|$ — число перестановок множества $\{1, 2, \dots, n\}$, которые имеют ровно k перемежающихся серий. Найдите рекуррентное соотношение, с помощью которого можно вычислить таблицу значений $\left|{n \atop k}\right|$; найдите также соответствующее рекуррентное соотношение для производящей функции $G_n(z) = \sum_k \left|{n \atop k}\right| z^k/n!$. Используя это последнее соотношение, найдите простую формулу для *дисперсии* числа перемежающихся серий в случайной перестановке множества $\{1, 2, \dots, n\}$.

17. [*M25*] Существует всего 2^n последовательностей $a_1 a_2 \dots a_n$, где каждый элемент a_j — либо 0, либо 1. Сколько среди них последовательностей, содержащих ровно k серий (т. е. содержащих ровно $k-1$ элементов a_j, таких, что $a_j > a_{j+1}$)?

18. [*M28*] Существует всего $n!$ последовательностей $b_1\, b_2 \ldots b_n$, в которых каждый элемент b_j — целое число, лежащее в диапазоне $0 \le b_j \le n - j$. Сколько среди них последовательностей, содержащих (a) ровно k нисходящих серий (т. е. содержащих ровно k соотношений элементов, таких, что $b_j > b_{j+1}$) и (b) ровно k отличающихся элементов?

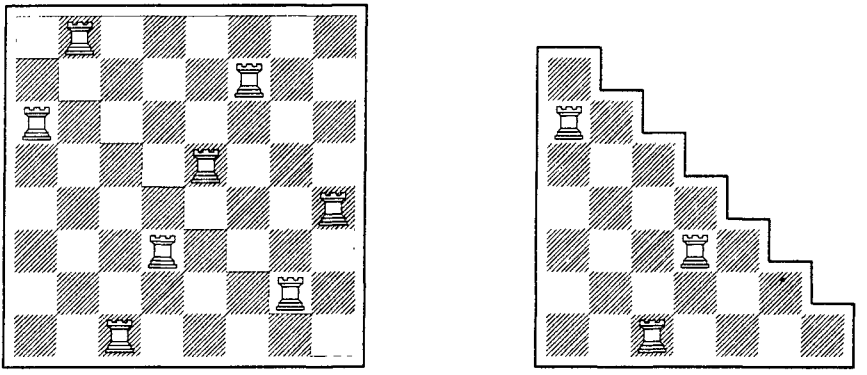


Рис. 4. Неатакующие ладьи на шахматной доске при заданном числе $k = 3$ ладей ниже главной диагонали.

▶ **19.** [*M26*] (Дж. Риордан (J. Riordan).) (a) Сколькими способами можно расположить n неатакующих ладей (т. е. никакие две ладьи не должны находиться на одной вертикали или горизонтали) на шахматной доске размером $n \times n$ так, чтобы ровно k из них находились на заданной стороне от главной диагонали? (b) Сколькими способами можно расположить k неатакующих ладей на заданной стороне от главной диагонали шахматной доски размером $n \times n$? Например, на рис. 4 показан один из 15 619 способов расположения восьми неатакующих ладей на обычной шахматной доске с тремя ладьями на незаштрихованном участке ниже главной диагонали, а также один из 1 050 способов расположения трех неатакующих ладей на треугольной доске.

▶ **20.** [*M21*] Говорят, что перестановка требует k *чтений*, если ее нужно просмотреть k раз слева направо, чтобы прочитать все элементы в порядке неубывания. Например, перестановка 4 9 1 8 2 5 3 6 7 требует четырех чтений: при первом чтении получаем 1, 2, 3, при втором — 4, 5, 6, 7; затем — 8 и 9. Найдите связь между сериями и чтениями.

21. [*M22*] Если перестановка $a_1\, a_2 \ldots a_n$ множества $\{1, 2, \ldots, n\}$ содержит k серий и требует j чтений в соответствии с упр. 20, то что можно сказать о перестановке $a_n \ldots a_2\, a_1$?

22. [*M26*] (Л. Карлиц, Д. П. Розель и Р. А. Скоувилл.) Покажите, что не существует перестановки множества $\{1, 2, \ldots, n\}$ с $n+1-r$ сериями, требующей s чтений, если $rs < n$; однако такая перестановка существует, если $n \ge n + 1 - r \ge s \ge 1$, $rs \ge n$.

23. [*HM42*] (Вальтер Вейссблюм (Walter Weissblum).) "Удлиненные серии" перестановки $a_1\, a_2 \ldots a_n$ получаются, если вставлять вертикальные черточки в тех местах, где нарушается установившаяся монотонность; удлиненные серии бывают как возрастающими, так и убывающими в зависимости от того, в каком порядке расположены первые два элемента, так что длина каждой удлиненной серии (кроме, возможно, последней) ≥ 2. Например, перестановка 7 5 | 6 2 | 3 8 9 | 1 4 содержит четыре удлиненные серии. Найдите средние длины первых двух удлиненных серий бесконечной перестановки и докажите, что в пределе длина удлиненной серии равна

$$(1 + \cot \tfrac{1}{2})/(3 - \cot \tfrac{1}{2}) \approx 2.4202.$$

24. [*M30*] Выразите в виде функции от p среднее число серий в последовательностях, полученных методом, который описан в упр. 5.1.1–18.

25. [*M25*] Пусть $U_1, \ldots, U_n$ — независимые равномерно распределенные числа на интервале $[0\,..\,1)$. Какова вероятность выполнения равенства $\lfloor U_1 + \cdots + U_n \rfloor = k$?

26. [*M20*] Обозначим через ϑ операцию $z\frac{d}{dz}$, которая умножает на n коэффициент при z^n в производящей функции. Покажите, что результат многократного (m раз) применения ϑ к $1/(1-z)$ может быть представлен в терминах чисел Эйлера.

▸ **27.** [*M21*] *Разрастающийся лес* — это лес, в котором узлы пронумерованы $\{1, 2, \ldots, n\}$ таким образом, что родители всегда имеют меньшие номера, чем их потомки. Покажите, что $\left\langle {n \atop k} \right\rangle$ — число n-узловых разрастающихся лесов, в которых имеется $k+1$ лист.

*5.1.4. Диаграммы и инволюции

В заключение нашего обзора комбинаторных свойств перестановок обсудим некоторые замечательные отношения, связывающие их с массивами целых чисел, называемыми диаграммами. *Диаграмма Юнга формы* $(n_1, n_2, \ldots, n_m)$, где $n_1 \geq n_2 \geq \cdots \geq n_m > 0$, — это расположение $n_1 + n_2 + \cdots + n_m$ различных целых чисел в массиве строк, выровненных по левому краю, где в i-й строке содержится n_i элементов; при этом в каждой строке элементы возрастают слева направо, а элементы каждого столбца возрастают сверху вниз. Например, диаграмма Юнга формы (6, 4, 4, 1) имеет вид

$$\begin{array}{|c|c|c|c|c|c|}\hline 1 & 2 & 5 & 9 & 10 & 15 \\ \hline 3 & 6 & 7 & 13 \\ \cline{1-4} 4 & 8 & 12 & 14 \\ \cline{1-4} 11 \\ \cline{1-1}\end{array} \quad . \tag{1}$$

Диаграммы такой формы ввел Альфред Юнг (Alfred Young) в 1900 году в качестве вспомогательного средства при изучении матричного представления перестановок. [См. *Proc. London Math. Soc.* (2) **28** (1928), 255–292; Bruce E. Sagan, *The Symmetric Group* (Pacific Grove, Calif.: Wadsworth & Brooks/Cole, 1991).] Для краткости вместо "диаграмма Юнга" мы будем говорить просто "диаграмма". *Инволюция* — это перестановка, обратная самой себе. Например, существует 10 инволюций множества $\{1, 2, 3, 4\}$:

$$\begin{pmatrix}1&2&3&4\\1&2&3&4\end{pmatrix} \begin{pmatrix}1&2&3&4\\2&1&3&4\end{pmatrix} \begin{pmatrix}1&2&3&4\\3&2&1&4\end{pmatrix} \begin{pmatrix}1&2&3&4\\4&2&3&1\end{pmatrix} \begin{pmatrix}1&2&3&4\\1&3&2&4\end{pmatrix}$$
$$\begin{pmatrix}1&2&3&4\\1&4&3&2\end{pmatrix} \begin{pmatrix}1&2&3&4\\1&2&4&3\end{pmatrix} \begin{pmatrix}1&2&3&4\\2&1&4&3\end{pmatrix} \begin{pmatrix}1&2&3&4\\3&4&1&2\end{pmatrix} \begin{pmatrix}1&2&3&4\\4&3&2&1\end{pmatrix} \quad . \tag{2}$$

Термин "инволюция" первоначально использовался в классических задачах геометрии; инволюции в общем смысле, рассматриваемые здесь, были впервые изучены Х. А. Роте (H. A. Rothe), когда он ввел понятие обратной перестановки (см. раздел 5.1.1).

Может показаться странным, что мы рассматриваем диаграммы и инволюции вместе, но существует удивительная связь между этими понятиями, не имеющими, казалось бы, друг к другу никакого отношения: *число инволюций множества*

$\{1, 2, \ldots, n\}$ *равно числу диаграмм, которые можно сформировать из элементов* $\{1, 2, \ldots, n\}$. Например, из $\{1, 2, 3, 4\}$ можно сформировать ровно 10 диаграмм:

$$\begin{array}{|c|c|c|c|}\hline 1&2&3&4\\\hline\end{array}\quad \begin{array}{|c|c|c|}\hline 1&3&4\\\hline \multicolumn{1}{|c|}{2}\\\cline{1-1}\end{array}\quad \begin{array}{|c|c|}\hline 1&4\\\hline \multicolumn{1}{|c|}{2}\\\cline{1-1} \multicolumn{1}{|c|}{3}\\\cline{1-1}\end{array}\quad \begin{array}{|c|c|}\hline 1&3\\\hline \multicolumn{1}{|c|}{2}\\\cline{1-1} \multicolumn{1}{|c|}{4}\\\cline{1-1}\end{array}\quad \begin{array}{|c|c|c|}\hline 1&2&4\\\hline \multicolumn{1}{|c|}{3}\\\cline{1-1}\end{array}$$

$$\begin{array}{|c|c|}\hline 1&2\\\hline \multicolumn{1}{|c|}{3}\\\cline{1-1} \multicolumn{1}{|c|}{4}\\\cline{1-1}\end{array}\quad \begin{array}{|c|c|c|}\hline 1&2&3\\\hline \multicolumn{1}{|c|}{4}\\\cline{1-1}\end{array}\quad \begin{array}{|c|c|}\hline 1&3\\\hline 2&4\\\hline\end{array}\quad \begin{array}{|c|c|}\hline 1&2\\\hline 3&4\\\hline\end{array}\quad \begin{array}{|c|}\hline 1\\\hline 2\\\hline 3\\\hline 4\\\hline\end{array} \tag{3}$$

Это соответствует 10 инволюциям (2). Такая связь между инволюциями и диаграммами отнюдь не очевидна, и, по-видимому, не существует простого способа ее доказательства. Доказательство, которое мы обсудим, включает интересный алгоритм построения диаграмм, обладающий и другими неожиданными свойствами; он основан на особой процедуре вставки в диаграмму новых элементов.

Предположим, например, что нужно вставить элемент 8 в диаграмму

$$\begin{array}{|c|c|c|c|c|c|}\hline 1&3&5&9&12&16\\\hline 2&6&10&15\\\cline{1-4} 4&13&14\\\cline{1-3} 11\\\cline{1-1} 17\\\cline{1-1}\end{array} \tag{4}$$

Метод, которым мы будем пользоваться, состоит в том, чтобы сначала поместить 8 в 1-ю строку, в клетку, ранее занимаемую 9, поскольку 9 — наименьший из элементов этой строки, больших 8. Элемент 9 "вытесняется" вниз, в строку 2, где он замещает 10. Затем 10 "вытесняет" 13 из 3-й строки в 4-ю и, поскольку в 4-й строке нет большего элемента, чем 13, процесс завершается добавлением 13 в правый конец строки 4. Наша диаграмма (4) преобразовалось в

$$\begin{array}{|c|c|c|c|c|c|}\hline 1&3&5&8&12&16\\\hline 2&6&9&15\\\cline{1-4} 4&10&14\\\cline{1-3} 11&13\\\cline{1-2} 17\\\cline{1-1}\end{array}\ . \tag{5}$$

В алгоритме I содержится точное описание этого процесса и доказательство того, что он всегда сохраняет свойства диаграммы.

Алгоритм I (*Вставка в диаграмму*). Пусть $P = (P_{ij})$ — диаграмма целых положительных чисел, а x — целое положительное число, не содержащееся в P. Этот алгоритм преобразует P в другую диаграмму, содержащую x наряду с исходными элементами P. Новая диаграмма имеет ту же форму, что и старая, но на пересечении строки s и столбца t появился новый элемент, где s и t — величины, определяемые алгоритмом.

(В этом алгоритме замечания в круглых скобках предназначены для доказательства его правильности, поскольку по индукции легко проверить, что эти замечания справедливы и что массив P продолжает оставаться диаграммой на протяжении всего процесса. Для удобства будем предполагать, что диаграмма ограничена нулями сверху и слева и символами "∞" — справа и снизу, так что элемент P_{ij} определен при всех $i, j \geq 0$. Если ввести отношение

$$\begin{gathered} a \lesssim b \quad \text{тогда и только тогда, когда} \\ a < b \quad \text{или} \quad a = b = 0, \quad \text{или} \quad a = b = \infty, \end{gathered} \tag{6}$$

то неравенства для диаграммы можно выразить в удобной форме:

$$\begin{gathered} P_{ij} = 0 \quad \text{тогда и только тогда, когда} \quad i = 0 \quad \text{или} \quad j = 0; \\ P_{ij} \lesssim P_{i(j+1)} \quad \text{и} \quad P_{ij} \lesssim P_{(i+1)j} \quad \text{при всех } i, j \geq 0. \end{gathered} \tag{7}$$

Утверждение "$x \notin P$" означает, что либо $x = \infty$, либо $x \neq P_{ij}$ ни при каких $i, j \geq 0$.)

I1. [Ввести x.] Присвоить $i \leftarrow 1$, $x_1 \leftarrow x$ и присвоить j наименьшее значение, такое, что $P_{1j} = \infty$.

I2. [Найти x_{i+1}.] (В этот момент $P_{(i-1)j} < x_i < P_{ij}$ и $x_i \notin P$.) Если $x_i < P_{i(j-1)}$, то уменьшить j на 1 и повторить этот шаг. В противном случае присвоить $x_{i+1} \leftarrow P_{ij}$ и $r_i \leftarrow j$.

I3. [Заменить элемеитом x_i.] (Теперь $P_{i(j-1)} < x_i < x_{i+1} = P_{ij} \lesssim P_{i(j+1)}$, $P_{(i-1)j} < x_i < x_{i+1} = P_{ij} \lesssim P_{(i+1)j}$ и $r_i = j$.) Присвоить $P_{ij} \leftarrow x_i$.

I4. [$x_{i+1} = \infty$?] (Теперь $P_{i(j-1)} < P_{ij} = x_i < x_{i+1} \lesssim P_{i(j+1)}$, $P_{(i-1)j} < P_{ij} = x_i < x_{i+1} \lesssim P_{(i+1)j}$, $r_i = j$ и $x_{i+1} \notin P$.) Если $x_{i+1} \neq \infty$, то увеличить i на 1 и вернуться к шагу I2.

I5. [Определить s, t.] Присвоить $s \leftarrow i$, $t \leftarrow j$ и закончить процедуру. (К этому моменту условия

$$P_{st} \neq \infty \quad \text{и} \quad P_{(s+1)t} = P_{s(t+1)} = \infty \tag{8}$$

будут выполнены.) ▮

Заметим, что данный алгоритм определяет "последовательность вытеснений"

$$x = x_1 < x_2 < \cdots < x_s < x_{s+1} = \infty, \tag{9}$$

а также вспомогательную последовательность индексов столбцов

$$r_1 \geq r_2 \geq \cdots \geq r_s = t; \tag{10}$$

при этом элемент P_{ir_i} меняет свое значение x_{i+1} на x_i, $1 \leq i \leq s$. Например, когда в диаграмму (4) вставлялся элемент 8, последовательность вытеснения имела вид 8, 9, 10, 13, ∞, а вспомогательная последовательность — 4, 3, 2, 2. Мы могли бы переформулировать алгоритм так, чтобы он расходовал меньше временной памяти (необходимо хранить только текущие значения j, x_i и x_{i+1}); последовательности (9) и (10) были введены с той лишь целью, чтобы можно было доказать некоторые интересные особенности этого алгоритма.

Основное свойство алгоритма I, которым мы не преминем воспользоваться, состоит в том, что алгоритм можно "прокрутить" в обратном направлении: по заданным s и t, определенным на шаге I5, можно привести P к исходному виду, найдя и удалив элемент x, который был вставлен. Рассмотрим, например, (5); пусть известно, что элемент 13 занимает позицию, которая раньше была пуста. Тогда элемент 13, наверное, был вытеснен вниз из 3-й строки числом 10, потому что 10 — наибольший элемент в этой строке, меньший 13; аналогично элемент 10, по-видимому, был вытеснен из 2-й строки числом 9, которое, в свою очередь, было вытеснено из 1-й строки числом 8. Таким образом, от (5) можно вернуться к (4). В следующем алгоритме подробно описан этот процесс.

Алгоритм D (*Удаление из диаграммы*). По заданной диаграмме P и целым положительным числам s, t, удовлетворяющим условиям (8), этот алгоритм преобразует P в другую диаграмму почти такой же формы, но с ∞ на пересечении столбца t и строки s. Найденный при этом элемент x удаляется из диаграммы P.

(Как и в алгоритме I, пояснения в круглых скобках включены для облегчения доказательства того, что массив P продолжает оставаться диаграммой на протяжении всего процесса.)

D1. [Ввести s, t.] Присвоить $j \leftarrow t$, $i \leftarrow s$, $x_{s+1} \leftarrow \infty$.

D2. [Найти x_i.] (В этот момент $P_{ij} < x_{i+1} \lesssim P_{(i+1)j}$ и $x_{i+1} \notin P$.) Если $P_{i(j+1)} < x_{i+1}$, то увеличить j на 1 и повторить этот шаг. В противном случае присвоить $x_i \leftarrow P_{ij}$ и $r_i \leftarrow j$.

D3. [Заменить элементом x_{i+1}.] (Теперь $P_{i(j-1)} < P_{ij} = x_i < x_{i+1} \lesssim P_{i(j+1)}$, $P_{(i-1)j} < P_{ij} = x_i < x_{i+1} \lesssim P_{(i+1)j}$ и $r_i = j$.) Присвоить $P_{ij} \leftarrow x_{i+1}$.

D4. [$i = 1$?] (Теперь $P_{i(j-1)} < x_i < x_{i+1} = P_{ij} \lesssim P_{i(j+1)}$, $P_{(i-1)j} < x_i < x_{i+1} = P_{ij} \lesssim P_{(i+1)j}$ и $r_i = j$.) Если $i > 1$, то уменьшить i на 1 и вернуться к шагу D2.

D5. [Определить x.] Присвоить $x \leftarrow x_1$ и закончить процедуру. (Теперь $0 < x < \infty$.) ▌

Пояснения к алгоритмам I и D, заключенные в круглые скобки, не только полезны для доказательства того факта, что эти алгоритмы сохраняют структуру диаграммы, но и позволяют убедиться, что *алгоритмы I и D взаимно обратны.* Если сначала выполнить алгоритм I с данной диаграммой P и некоторым целым положительным числом $x \notin P$, то он вставит x в P и определит целые положительные числа s, t, удовлетворяющие условиям (8). Алгоритм D, примененный к полученному результату, восстановит значения x и первоначальный вид P. Обратно, если сначала выполнить алгоритм D с данной диаграммой P и некоторыми целыми положительными числами s, t, удовлетворяющими условиям (8), то он модифицирует P, удалив некоторое целое положительное число x. Алгоритм I, примененный к полученному результату, восстановит значения s, t и первоначальный вид P. Дело в том, что присвоения, приведенные в заключенных в скобки пояснениях к шагам I3 и D4, равно как и к шагам I4 и D3, совпадают и однозначно определяют значение j. Следовательно, вспомогательные последовательности (9), (10) одинаковы в обоих случаях.

Теперь мы подготовлены к доказательству основного свойства диаграммы.

Теорема А. *Существует взаимно однозначное соответствие между множеством всех перестановок множества* $\{1, 2, \ldots, n\}$ *и множеством всех упорядоченных пар диаграмм* (P, Q), *где* P *и* Q *— диаграммы одинаковой формы из элементов* $\{1, 2, \ldots, n\}$.

(Пример к этой теореме содержится в приведенном ниже доказательстве.)

Доказательство. Удобнее доказать несколько более общий результат. По произвольному двухстрочному массиву

$$\begin{pmatrix} q_1 & q_2 & \ldots & q_n \\ p_1 & p_2 & \ldots & p_n \end{pmatrix}, \qquad \begin{array}{l} q_1 < q_2 < \cdots < q_n, \\ p_1, p_2, \ldots, p_n \text{ различны}, \end{array} \tag{11}$$

построим две соответствующие диаграммы P и Q, где P состоит из элементов $\{p_1, \ldots, p_n\}$, а Q — из элементов $\{q_1, \ldots, q_n\}$, причем P и Q имеют одинаковую форму.

Пусть P и Q вначале пусты. При $i = 1, 2, \ldots, n$ (именно в таком порядке) выполним следующую операцию: вставим p_i в диаграмму P при помощи алгоритма I; затем установим $Q_{st} \leftarrow q_i$, где s и t определяют вновь заполненную позицию в P.

Например, если задана перестановка $\left(\begin{smallmatrix} 1 & 3 & 5 & 6 & 8 \\ 7 & 2 & 9 & 5 & 3 \end{smallmatrix}\right)$, действуем следующим образом:

	P	Q
Вставка 7:	$\begin{array}{\|c\|}\hline 7 \\ \hline\end{array}$	$\begin{array}{\|c\|}\hline 1 \\ \hline\end{array}$
Вставка 2:	$\begin{array}{\|c\|}\hline 2 \\ \hline 7 \\ \hline\end{array}$	$\begin{array}{\|c\|}\hline 1 \\ \hline 3 \\ \hline\end{array}$
Вставка 9:	$\begin{array}{\|c\|c\|}\hline 2 & 9 \\ \hline 7 \\ \cline{1-1}\end{array}$	$\begin{array}{\|c\|c\|}\hline 1 & 5 \\ \hline 3 \\ \cline{1-1}\end{array}$.
Вставка 5:	$\begin{array}{\|c\|c\|}\hline 2 & 5 \\ \hline 7 & 9 \\ \hline\end{array}$	$\begin{array}{\|c\|c\|}\hline 1 & 5 \\ \hline 3 & 6 \\ \hline\end{array}$
Вставка 3:	$\begin{array}{\|c\|c\|}\hline 2 & 3 \\ \hline 5 & 9 \\ \hline 7 \\ \cline{1-1}\end{array}$	$\begin{array}{\|c\|c\|}\hline 1 & 5 \\ \hline 3 & 6 \\ \hline 8 \\ \cline{1-1}\end{array}$

(12)

Значит, пара диаграмм (P, Q), соответствующая перестановке $\left(\begin{smallmatrix} 1 & 3 & 5 & 6 & 8 \\ 7 & 2 & 9 & 5 & 3 \end{smallmatrix}\right)$, имеет вид

$$P = \begin{array}{|c|c|}\hline 2 & 3 \\ \hline 5 & 9 \\ \hline 7 \\ \cline{1-1}\end{array}, \qquad Q = \begin{array}{|c|c|}\hline 1 & 5 \\ \hline 3 & 6 \\ \hline 8 \\ \cline{1-1}\end{array}. \tag{13}$$

Из построения ясно, что P и Q всегда имеют одну форму. Кроме того, поскольку элементы всегда добавляются на границу Q и в порядке возрастания, то Q — диаграмма.

Обратно, если заданы две диаграммы одинаковой формы, то соответствующий двухстрочный массив (11) можно построить так. Пусть элементами Q являются

$$q_1 < q_2 < \cdots < q_n.$$

Пусть также p_i есть элемент x, который удаляется из P по алгоритму D с использованием значений s и t, таких, что $Q_{st} = q_i$, причем $i = n, \ldots, 2, 1$ (именно в таком порядке).

Например, если применить это построение к диаграмме (13) и производить вычисления, обратные (12), до тех пор, пока P не исчерпается, то получится массив $\left(\begin{smallmatrix} 1 & 3 & 5 & 6 & 8 \\ 7 & 2 & 9 & 5 & 3 \end{smallmatrix}\right)$.

Поскольку алгоритмы I и D взаимно обратны, взаимно обратны и описанные здесь два построения; таким образом, требуемое взаимно однозначное соответствие установлено. ▌

Соответствие, определенное в доказательстве теоремы A, обладает множеством поразительных свойств, и теперь мы приступим к выводу некоторых из них. Убедительная просьба к читателю: прежде чем двигаться дальше, выполните упр. 1, чтобы освоить методику построений.

Как только элемент вытеснен из строки 1 в строку 2, он уже не влияет на строку 1; кроме того, строки 2, 3, ... строятся из последовательности "вытесненных" элементов так же, как строки 1, 2, ... строятся из исходной перестановки. Это наводит на мысль о том, что на построение в теореме A можно взглянуть иначе, обращая внимание лишь на первые строки P и Q. Например, перестановка $\left(\begin{smallmatrix} 1 & 3 & 5 & 6 & 8 \\ 7 & 2 & 9 & 5 & 3 \end{smallmatrix}\right)$ вызывает следующие действия над строкой 1 [ср. с (12)]:

$$\begin{array}{ll} 1: & \text{Вставить 7, присвоить } Q_{11} \leftarrow 1. \\ 3: & \text{Вставить 2, вытеснить 7.} \\ 5: & \text{Вставить 9, присвоить } Q_{12} \leftarrow 5. \\ 6: & \text{Вставить 5, вытеснить 9.} \\ 8: & \text{Вставить 3, вытеснить 5.} \end{array} \tag{14}$$

Таким образом, первая строка P — это 2 3, а первая строка Q — это 1 5. Кроме того, остальные строки P и Q составляют диаграммы, соответствующие "вытесненному" двухстрочному массиву

$$\begin{pmatrix} 3 & 6 & 8 \\ 7 & 9 & 5 \end{pmatrix}. \tag{15}$$

Чтобы понять, как строится строка 1, проанализируем элементы, попадающие в некоторый заданный столбец этой строки. Будем говорить, что пара (q_i, p_i) принадлежит *классу* t двухстрочного массива

$$\begin{pmatrix} q_1 & q_2 & \cdots & q_n \\ p_1 & p_2 & \cdots & p_n \end{pmatrix}, \qquad \begin{array}{l} q_1 < q_2 < \cdots < q_n, \\ p_1, p_2, \ldots, p_n \text{ различны,} \end{array} \tag{16}$$

если после применения алгоритма I последовательно к $p_1, p_2, \ldots, p_i$, начиная с пустой диаграммы P, оказывается, что $p_i = P_{1t}$. (Напомним, что алгоритм I всегда вставляет данный элемент в 1-ю строку.)

Легко видеть, что (q_i, p_i) принадлежит классу 1 тогда и только тогда, когда p_i имеет $i - 1$ инверсий, т. е. тогда и только тогда, когда $p_i = \min\{p_1, p_2, \ldots, p_i\}$ — "левосторонний минимум". Если в массиве (16) вычеркнуть столбцы класса 1, то получится двухстрочный массив

$$\begin{pmatrix} q'_1 & q'_2 & \cdots & q'_m \\ p'_1 & p'_2 & \cdots & p'_m \end{pmatrix}, \tag{17}$$

такой, что пара (q,p) принадлежит классу t относительно (17) тогда и только тогда, когда она принадлежит классу $t+1$ относительно массива (16). Операция перехода от (16) к (17) соответствует удалению крайней слева позиции строки 1. Это дает систематический способ определения классов. Например, в перестановке $\binom{1\ 3\ 5\ 6\ 8}{7\ 2\ 9\ 5\ 3}$ левосторонними минимумами являются элементы 7 и 2, так что класс 1 — это $\{(1,7),(3,2)\}$; в оставшемся массиве $\binom{5\ 6\ 8}{9\ 5\ 3}$ все элементы минимальны, так что класс 2 — это $\{(5,9),(6,5),(8,3)\}$. В "вытесненном" массиве (15) класс 1 — это $\{(3,7),(8,5)\}$, а класс 2 — $\{(6,9)\}$.

Для любого фиксированного t элементы класса t можно так пометить

$$(q_{i_1},p_{i_1}),\ldots,(q_{i_k},p_{i_k}),$$

чтобы выполнялись неравенства

$$\begin{aligned} q_{i_1} &< q_{i_2} < \cdots < q_{i_k}, \\ p_{i_1} &> p_{i_2} > \cdots > p_{i_k}, \end{aligned} \tag{18}$$

поскольку в процессе выполнения алгоритма вставки позиция диаграммы P_{1t} принимает убывающую последовательность значений $p_{i_1},\ldots,p_{i_k}$. В конце построения

$$P_{1t} = p_{i_k}, \qquad Q_{1t} = q_{i_1}, \tag{19}$$

а вытесненный двухстрочный массив, которым определяются строки 2, 3, ... диаграмм P и Q, содержит столбцы

$$\begin{pmatrix} q_{i_2} & q_{i_3} & \cdots & q_{i_k} \\ p_{i_1} & p_{i_2} & \cdots & p_{i_{k-1}} \end{pmatrix} \tag{20}$$

и другие столбцы, аналогичным образом полученные из других классов.

Эти рассуждения приводят нас к простому методу вычисления P и Q вручную (см. упр. 3), а также предоставляют средства для доказательства одного весьма неожиданного результата.

Теорема В. *Если в построении из теоремы А перестановка*

$$\begin{pmatrix} 1 & 2 & \ldots & n \\ a_1 & a_2 & \ldots & a_n \end{pmatrix}$$

соответствует диаграмме (P,Q), то обратная ей перестановка соответствует диаграмме (Q,P).

Это довольно удивительный факт, потому что в теореме А диаграммы P и Q формируются совершенно разными способами и обратная перестановка получается в результате весьма причудливой перетасовки столбцов двухстрочного массива.

Доказательство. Предположим, имеется двухстрочный массив (16); поменяв местами его строки и рассортировав столбцы так, чтобы элементы новой верхней строки расположились в порядке неубывания, получим "обратный" массив

$$\begin{aligned} \begin{pmatrix} q_1 & q_2 & \ldots & q_n \\ p_1 & p_2 & \ldots & p_n \end{pmatrix}^{-} &= \begin{pmatrix} p_1 & p_2 & \ldots & p_n \\ q_1 & q_2 & \ldots & q_n \end{pmatrix} \\ &= \begin{pmatrix} p'_1 & p'_2 & \ldots & p'_n \\ q'_1 & q'_2 & \ldots & q'_n \end{pmatrix}, \quad \begin{aligned} &p'_1 < p'_2 < \cdots < p'_n; \\ &q'_1, q'_2, \ldots, q'_n \text{ различны.} \end{aligned} \end{aligned} \tag{21}$$

Покажем, что эта операция соответствует взаимной замене P и Q в построении из теоремы А.

В упр. 2 наши замечания об определении классов переформулированы таким образом, что класс, к которому относится пара (q_i, p_i), не зависит от того факта, что элементы $q_1, q_2, \ldots, q_n$ расположены в порядке возрастания. Поскольку результирующие условия симметричны относительно q и p, операция (21) не нарушает структуру классов; если (q, p) принадлежит классу t относительно (16), то (p, q) принадлежит классу t относительно (21). Поэтому, если разместить элементы последнего класса t так, чтобы

$$\begin{aligned} p_{i_k} &< \cdots < p_{i_2} < p_{i_1}, \\ q_{i_k} &> \cdots > q_{i_2} > q_{i_1}, \end{aligned} \tag{22}$$

то по аналогии с (18) получим

$$P_{1t} = q_{i_1}, \qquad Q_{1t} = p_{i_k}, \tag{23}$$

как в (19), а столбцы

$$\begin{pmatrix} p_{i_{k-1}} & \cdots & p_{i_2} & p_{i_1} \\ q_{i_k} & \cdots & q_{i_3} & q_{i_2} \end{pmatrix} \tag{24}$$

войдут в вытесненный массив, как в (20). Следовательно, первые строки P и Q меняются местами. Кроме того, вытесненный двухстрочный массив для (21) является обратным по отношению к вытесненному двухстрочному массиву для (16), так что доказательство завершается применением индукции по числу строк в диаграмме. ▌

Следствие. *Количество диаграмм, которые можно сформировать из элементов* $\{1, 2, \ldots, n\}$, *равно количеству инволюций множества* $\{1, 2, \ldots, n\}$.

Доказательство. Если π — инволюция, соответствующая паре диаграмм (P, Q), то $\pi = \pi^-$ соответствует паре (Q, P). Следовательно, $P = Q$. Обратно, если π — какая-либо перестановка, соответствующая паре (P, P), то π^- тоже соответствует паре (P, P); отсюда $\pi = \pi^-$. Таким образом, существует взаимно однозначное соответствие между инволюциями π и диаграммой P. ▌

Ясно, что элемент в левом верхнем углу диаграммы всегда наименьший. Это наводит на мысль о возможном способе сортировки множества чисел. Сначала можно составить из них диаграмму, многократно применяя алгоритм I, и в результате наименьший элемент окажется в углу. Затем этот наименьший элемент удаляется, а остальные элементы переразмещаются так, чтобы образовалась другая диаграмма; потом удаляется новый минимальный элемент и т. д.

Поэтому давайте посмотрим, что происходит, когда мы удаляем угловой элемент из диаграммы

$$\begin{array}{|c|c|c|c|c|c|} \hline 1 & 3 & 5 & 7 & 11 & 15 \\ \hline 2 & 6 & 8 & 14 \\ \cline{1-4} 4 & 9 & 13 \\ \cline{1-3} 10 & 12 \\ \cline{1-2} 16 \\ \cline{1-1} \end{array} \quad . \tag{25}$$

После удаления 1 на освободившееся место необходимо поставить 2. Затем можно поднять 4 на место 2, однако 10 нельзя поднять на место 4; на это место можно

подвинуть 9, а потом 12 — на место 9. В общем случае приходим к следующей процедуре.

Алгоритм S (*Удаление углового элемента*). Этот алгоритм удаляет элемент из левого верхнего угла диаграммы P и перемещает остальные элементы так, чтобы сохранились свойства диаграммы. Далее используются те же обозначения, что и в алгоритмах I и D.

S1. [Начальиая установка.] Присвоить $r \leftarrow 1$, $s \leftarrow 1$.

S2. [Выполнено?] Если $P_{rs} = \infty$, то процесс завершен.

S3. [Сравнить.] Если $P_{(r+1)s} \lesssim P_{r(s+1)}$, то перейти к шагу S5. (Сравниваем элементы справа и снизу от свободного места и передвигаем меньший из них.)

S4. [Сдвиг влево.] Присвоить $P_{rs} \leftarrow P_{r(s+1)}$, $s \leftarrow s+1$ и вернуться к шагу S3.

S5. [Сдвиг вверх.] Присвоить $P_{rs} \leftarrow P_{(r+1)s}$, $r \leftarrow r+1$ и вернуться к шагу S2. ▮

Легко доказать, что после удаления с помощью алгоритма S углового элемента P по-прежнему является диаграммой (см. упр. 10). Таким образом, применяя алгоритм S до тех пор, пока P не исчерпается, можно прочитать его элементы в порядке возрастания. К сожалению, этот алгоритм сортировки оказывается не столь эффективным, как другие алгоритмы, которые нам еще предстоит рассмотреть. Минимальное время его работы пропорционально $n^{1.5}$; аналогичные алгоритмы, использующие вместо диаграмм деревья, затрачивают время порядка $n \log n$.

Хотя алгоритм S в приложении к сортировке и не отличается особой эффективностью, он обладает некоторыми очень интересными свойствами.

Теорема C (М. П. Шуценберже (M. P. Schützenberger)). *Если P — диаграмма, построенная, как в теореме A, из перестановки $a_1\, a_2 \ldots a_n$, и если*

$$a_i = \min\{a_1, a_2, \ldots, a_n\},$$

то алгоритм S преобразует P в диаграмму, соответствующую перестановке $a_1 \ldots a_{i-1}\, a_{i+1} \ldots a_n$.

Доказательство. См. упр. 13. ▮

Давайте после применения алгоритма S поместим на вновь освободившееся место P_{rs} удаленный элемент, но отобразим его ***курсивом***, указав таким образом, что на самом деле элемент не является частью диаграммы. Применив, например, эту процедуру к диаграмме (25), мы получим

2	3	5	7	11	15
4	6	8	14		
9	12	13			
10	*1*				
16					

,

а выполнив такую же процедуру еще два раза, получим

4	5	7	11	15	*2*
6	8	13	14		
9	12	*3*			
10	*1*				
16					

.

Продолжая до тех пор, пока все элементы не будут удалены, получим конфигурацию

16	*14*	*13*	*12*	*10*	*2*
15	*9*	*6*	*4*		
11	*5*	*3*			
8	*1*				
7					

, (26)

имеющую ту же форму, что и исходная диаграмма (25). Эту конфигурацию можно назвать ***двойственной диаграммой***, потому что она похожа на диаграмму с той лишь разницей, что применяется "двойственное отношение порядка" ($>$ и $<$ поменялись ролями). Обозначим двойственную диаграмму, полученную из P таким способом, через P^S.

Диаграмма P определяется из P^S единственным образом. В самом деле, исходную диаграмму можно получить из P^S при помощи того же алгоритма, но изменив на обратные отношения порядка и роли обычных и представленных курсивом элементов, поскольку P^S — двойственная диаграмма. Например, применив к (26) два шага этого алгоритма, получим

14	*13*	*12*	*10*	*2*	15
11	*9*	*6*	*4*		
8	*5*	*3*			
7	*1*				
16					

.

В конце концов, опять получается диаграмма (25)! Этот замечательный результат — одно из следствий нашей следующей теоремы.

Теорема D (К. Шенстед (C. Schensted), М. П. Шуценберже (M. P. Schützenberger)). *Пусть*

$$\begin{pmatrix} q_1 & q_2 & \cdots & q_n \\ p_1 & p_2 & \cdots & p_n \end{pmatrix} \tag{27}$$

представляет собой двухстрочный массив, соответствующий паре диаграмм (P, Q).

а) *Если пользоваться двойственным (обратным) отношением порядка для* q, *но не для* p, *то двухстрочный массив*

$$\begin{pmatrix} q_n & \cdots & q_2 & q_1 \\ p_n & \cdots & p_2 & p_1 \end{pmatrix} \tag{28}$$

соответствует паре $\left(P^T, (Q^S)^T\right)$.

Как обычно, через "T" обозначена операция транспонирования строк и столбцов; P^T — диаграмма, а $(Q^S)^T$ — двойственная диаграмма, поскольку элементы q расположены в обратном порядке.

b) *Если пользоваться двойственным отношением порядка для p, но не для q, то двухстрочный массив* (27) *соответствует паре* $\big((P^S)^T, Q^T\big)$.

c) *Если пользоваться двойственным отношением порядка как для p, так и для q, то двухстрочный массив* (28) *соответствует паре* (P^S, Q^S).

Доказательство. Простое доказательство этой теоремы неизвестно. То, что случай (a) соответствует паре (P^T, X), где X — некоторая двойственная диаграмма, доказано в упр. 5; следовательно, по теореме B случай (b) соответствует паре (Y, Q^T) для некоторой двойственной диаграммы Y и Y должна иметь ту же форму, что и P^T.

Пусть $p_i = \min\{p_1, \ldots, p_n\}$; так как p_i — "наибольший" элемент при двойственном отношении порядка, то он оказывается на границе Y и не вытесняет никаких элементов при построении из теоремы A. Таким образом, если последовательно вставлять элементы $p_1, \ldots, p_{i-1}, p_{i+1}, \ldots, p_n$, применяя двойственное отношение порядка, то получится $Y - \{p_i\}$, т. е. Y, из которой удален элемент p_i. По теореме C, если последовательно вставлять элементы $p_1, \ldots, p_{i-1}, p_{i+1}, \ldots, p_n$, применяя обычное отношение порядка, построим диаграмму $d(P)$, которая получается путем применения к P алгоритма S. Индукция по n дает $Y - \{p_i\} = \big(d(P)^S\big)^T$. Но поскольку

$$(P^S)^T - \{p_i\} = \big(d(P)^S\big)^T \tag{29}$$

по определению операции S, а Y имеет ту же форму, что и $(P^S)^T$, должно иметь место равенство $Y = (P^S)^T$.

Тем самым доказано утверждение (b); (a) получается в результате применения теоремы B. Последовательное применение (a) и (b) показывает, что случай (c) соответствует паре $\big(((P^T)^S)^T, ((Q^S)^T)^T\big)$, а это равно (P^S, Q^S), так как $(P^S)^T = (P^T)^S$ вследствие симметрии операции S по отношению к строкам и столбцам. ▌

Эта теорема, в частности, устанавливает два удивительных факта, касающихся алгоритма вставки в диаграмму. Если в результате последовательной вставки различных элементов $p_1, \ldots, p_n$ в пустую диаграмму получается диаграмма P, то в результате вставки этих элементов в обратном порядке, $p_n, \ldots, p_1$, получится *транспонированная* диаграмма P^T. Если же мы не только станем вставлять элементы в порядке $p_n, \ldots, p_1$, но и поменяем ролями $>$ и $<$, а также 0 и ∞, то получим двойственную диаграмму P^S. Настоятельно рекомендуем читателю проанализировать эти процессы самостоятельно на нескольких простых примерах. Необычная природа этих совпадений может вызвать подозрение о вмешательстве каких-то потусторонних сил. До сих пор неизвестно какое-либо простое объяснение подобных явлений; кажется, не существует простого способа доказать даже то, что случай (c) соответствует диаграмме той же *формы*, что и P и Q, хотя метод характеристики классов, использованный в упр. 2, может послужить отправной точкой для дальнейших поисков.

Соответствие, устанавливаемое теоремой A, было найдено Ж. Б. Робинсоном (G. de B. Robinson) [*American J. Math.* **60** (1938), 745–760, §5] в несколько иной

и довольно туманной форме как часть решения весьма сложной задачи из теории групп. Он сформулировал теорему B без доказательства. Много лет спустя K. Шенстед независимо заново открыл это соответствие, которое сформулировал в терминах "вытеснения", т. е., по существу, в той же форме, которую использовали мы в алгоритме I. Шенстед также доказал часть теоремы D (a), касающуюся "P" [см. *Canadian J. Math.* **13** (1961), 179–191]. В работе М. П. Шуценберже [*Math. Scand.* **12** (1963), 117–128] доказана теорема B и "Q"-часть теоремы D (a), из которой следуют (b) и (c). Это соответствие можно распространить и на перестановки *мультимножеств;* случай, когда $p_1, \ldots, p_n$ необязательно различны, рассмотрел Шенстед, а "максимальное" обобщение, когда и p, и q могут содержать повторяющиеся элементы, исследовано Кнутом [*Pacific J. Math.* **34** (1970), 709–727].

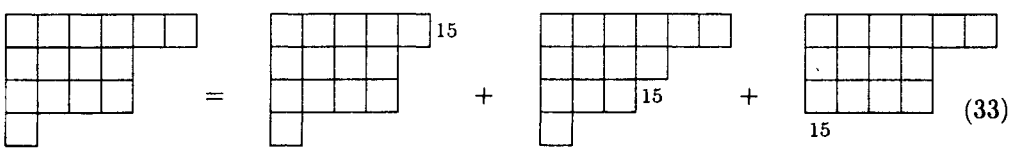
Обратимся теперь к родственному вопросу: *сколько диаграмм, составленных из элементов* $\{1, 2, \ldots, n\}$, *имеют данную форму* $(n_1, n_2, \ldots, n_m)$, *где* $n_1 + n_2 + \cdots + n_m = n$? Обозначим это число через $f(n_1, n_2, \ldots, n_m)$; если параметры n_j — произвольные целые числа, то функция f должна удовлетворять соотношениям

$$f(n_1, n_2, \ldots, n_m) = 0, \qquad \text{если не} \qquad n_1 \geq n_2 \geq \cdots \geq n_m \geq 0; \tag{30}$$

$$f(n_1, n_2, \ldots, n_m, 0) = f(n_1, n_2, \ldots, n_m); \tag{31}$$

$$\begin{aligned} f(n_1, n_2, \ldots, n_m) = {} & f(n_1 - 1, n_2, \ldots, n_m) + f(n_1, n_2 - 1, \ldots, n_m) \\ & + \cdots + f(n_1, n_2, \ldots, n_m - 1), \\ & \text{если} \quad n_1 \geq n_2 \geq \cdots \geq n_m \geq 1. \end{aligned} \tag{32}$$

Рекуррентное соотношение (32) вытекает из того факта, что при удалении из диаграммы наибольшего элемента всегда снова получается диаграмма; например, количество диаграмм формы $(6, 4, 4, 1)$ равно $f(5, 4, 4, 1) + f(6, 3, 4, 1) + f(6, 4, 3, 1) + f(6, 4, 4, 0) = f(5, 4, 4, 1) + f(6, 4, 3, 1) + f(6, 4, 4)$, потому что всякая диаграмма формы $(6, 4, 4, 1)$ из элементов $\{1, 2, \ldots, 15\}$ получается в результате вставки элемента 15 в подходящее место в диаграмме формы $(5, 4, 4, 1)$, $(6, 4, 3, 1)$ или $(6, 4, 4)$. Изобразим это схематично:



(33)

Функция $f(n_1, n_2, \ldots, n_m)$, удовлетворяющая таким соотношениям, имеет довольно простой вид:

$$f(n_1, n_2, \ldots, n_m) = \frac{\Delta(n_1 + m - 1,\, n_2 + m - 2,\, \ldots,\, n_m)\, n!}{(n_1 + m - 1)!\,(n_2 + m - 2)! \ldots n_m!}, \tag{34}$$

при условии, что соблюдается соотношение

$$n_1 + m - 1 \geq n_2 + m - 2 \geq \cdots \geq n_m.$$

Здесь через Δ обозначена функция "квадратный корень из дискриминанта"

$$\Delta(x_1, x_2, \ldots, x_m) = \det \begin{pmatrix} x_1^{m-1} & x_2^{m-1} & \ldots & x_m^{m-1} \\ \vdots & \vdots & & \vdots \\ x_1^2 & x_2^2 & & x_m^2 \\ x_1 & x_2 & & x_m \\ 1 & 1 & \ldots & 1 \end{pmatrix} = \prod_{1 \le i < j \le m} (x_i - x_j). \qquad (35)$$

Формулу (34) вывел Г. Фробениус [см. G. Frobenius *Sitzungsberichte preuß. Akad. der Wissenschaften* (1900), 516–534, §3], изучая эквивалентную задачу теории групп; он использовал довольно глубокую аргументацию, опирающуюся на теорию групп. Комбинаторное доказательство независимо нашел Мак-Магон [см. MacMahon, *Philosophical Trans.* **A209** (1909), 153–175]. Эту формулу можно доказать по индукции, так как (30) и (31) доказываются без труда, а формула (32) получается, если положить $y = -1$ в тождестве из упр. 17.

Из теоремы А в связи с этой формулой для числа диаграмм вытекает замечательное тождество. Взяв сумму по всевозможным формам диаграмм, получим

$$\begin{aligned} n! &= \sum_{\substack{k_1 \ge k_2 \ge \cdots \ge k_n \ge 0 \\ k_1 + k_2 + \cdots + k_n = n}} f(k_1, k_2 \ldots, k_n)^2 \\ &= n!^2 \sum_{\substack{k_1 \ge k_2 \ge \cdots \ge k_n \ge 0 \\ k_1 + k_2 + \cdots + k_n = n}} \frac{\Delta(k_1 + n - 1,\, k_2 + n - 2,\, \ldots,\, k_n)^2}{(k_1 + n - 1)!^2\, (k_2 + n - 2)!^2 \ldots k_n!^2} \\ &= n!^2 \sum_{\substack{q_1 > q_2 > \cdots > q_n \ge 0 \\ q_1 + q_2 + \cdots + q_n = (n+1)n/2}} \frac{\Delta(q_1, q_2, \ldots, q_n)^2}{q_1!^2\, q_2!^2 \ldots q_n!^2}; \end{aligned}$$

отсюда

$$\sum_{\substack{q_1 + q_2 + \cdots + q_n = (n+1)n/2 \\ q_1, q_2, \ldots, q_n \ge 0}} \frac{\Delta(q_1, q_2, \ldots, q_n)^2}{q_1!^2\, q_2!^2 \ldots q_n!^2} = 1. \qquad (36)$$

В последней сумме отсутствуют неравенства $q_1 > q_2 > \cdots > q_n$, потому что слагаемые — симметричные относительно q функции, которые обращаются в 0 при $q_i = q_j$. Аналогичное тождество появляется в упр. 24.

Формулу для числа диаграмм можно представить в более привлекательной форме, если ввести понятие "уголок". В *уголок*, соответствующий клетке диаграммы, входит сама эта клетка плюс все клетки, расположенные в той же колонке снизу и в той же строке справа от нее. Например, заштрихованный участок на рис. 5 — это уголок, соответствующий клетке $(2, 3)$ строки 2 и столбца 3; он состоит из шести клеток. В каждой клетке на рис. 5 записана длина соответствующего ей уголка.

Если диаграмма имеет форму $(n_1, n_2, \ldots, n_m)$, то длина самого длинного уголка равна $n_1 + m - 1$. Дальнейший анализ длин уголков показывает, что строка 1 содержит все длины $n_1 + m - 1$, $n_1 + m - 2$, ..., 1, *кроме* $(n_1 + m - 1) - (n_m)$, $(n_1 + m - 1) - (n_{m-1} + 1)$, ..., $(n_1 + m - 1) - (n_2 + m - 2)$. Например, на рис. 5 длины уголков в 1-й строке суть 12, 11, 10, ..., 1, за исключением 10, 9, 6, 3, 2; эти исключения соответствуют пяти несуществующим уголкам, начинающимся в несуществующих клетках

12	11	8	7	5	4	1
10	9	6	5	3	2	•
9	8	5	4	2	1	•
6	5	2	1	•		
3	2	•				
2	1	•				

Рис. 5. Уголки и длины уголков.

$(6,3)$, $(5,3)$, $(4,5)$, $(3,7)$, $(2,7)$ и заканчивающимся в клетке $(1,7)$. Аналогично j-я строка содержит все длины уголков n_j+m-j, ..., 1, кроме $(n_j+m-j)-(n_m)$, ..., $(n_j+m-j)-(n_{j+1}+m-j-1)$. Отсюда следует, что произведение длин всех уголков равно

$$\frac{(n_1+m-1)!\,(n_2+m-2)!\ldots n_m!}{\Delta(n_1+m-1,n_2+m-2,\ldots,n_m)}.$$

Это выражение входит в формулу (34). Итак, мы получили результат, который в работе J. S. Frame, G. de B. Robinson, R. M. Thrall, *Canadian J. Math.* **6** (1954), 316–324, сфомулирован в виде теоремы.

Теорема Н. *Количество диаграмм данной формы, составленных из элементов* $\{1,2,\ldots,n\}$, *равно* $n!$, *деленному на произведение длин уголков.* ▮

Такая лаконичная формулировка заслуживает и лаконичного доказательства. Мы приведем нестрогое рассуждение, основанное на эвристике. Каждый элемент диаграммы — минимальный в своем уголке; если заполнить клетки диаграммы случайным образом, то вероятность того, что в клетке (i,j) окажется минимальный элемент соответствующего уголка, есть величина, обратная длине уголка. Перемножение этих вероятностей по всем i и j приводит к формулировке теоремы Н. Однако такое рассуждение ошибочно, поскольку эти события отнюдь не являются независимыми! Все известные доказательства теоремы Н, основанные на комбинаторных свойствах уголков, были некорректны вплоть до 1992 года (см. упр. 39), хотя и предлагалось несколько непрямых доказательств (см. упр. 35, 36 и 38).

Существует интересная связь между теоремой Н и перечислением деревьев, которое рассматривалось в главе 2.Мы видели, что бинарным деревьям с n узлами соответствуют перестановки, которые можно получить с помощью стека, и что этим перестановкам соответствуют последовательности $a_1\,a_2\ldots a_{2n}$ n литер S и n литер·X, такие, что количество литер S никогда не бывает меньше количества литер X, если читать последовательность слева направо (см. упр. 2.2.1–3 и 2.3.1–6). Подобным последовательностям естественным образом сопоставляются диаграммы формы (n,n); в 1-ю строку помещаются индексы i, такие, что a_i = S, а во 2-ю строку — индексы, при которых a_i = X. Например, последовательности

S S S X X S S X X S X X

соответствует диаграмма

1	2	3	6	7	10
4	5	8	9	11	12

. (37)

Условие, налагаемое на столбцы, удовлетворяется в такой диаграмме в том и только том случае, когда при чтении слева направо число литер X никогда не превышает числа литер S. По теореме H количество всевозможных диаграмм формы (n, n) равно

$$\frac{(2n)!}{(n+1)!\,n!}\,;$$

следовательно, таково же число бинарных деревьев, что согласуется с формулой 2.3.4.4–(14). Более того, если воспользоваться диаграммой формы (n, m) при $n \geq m$, то при помощи этого рассуждения можно решить и более общую "задачу о баллотировке", рассмотренную в ответе к упр. 2.2.1–4. Таким образом, теорема H в качестве простых частных случаев включает в себя некоторые весьма сложные задачи о перечислении.

Всякой диаграмме A формы (n, n) из элементов $\{1, 2, \ldots, 2n\}$ соответствуют две диаграммы (P, Q) одинаковой формы. Следующий способ построения такого соответствия предложен Мак-Магоном [*Combinatory Analysis* **1** (1915), 130–131]. Пусть P состоит из элементов $\{1, \ldots, n\}$, расположенных так, как в A, а Q получается, если взять остальные элементы A, повернуть всю конфигурацию на 180° и заменить $n+1$, $n+2$, ..., $2n$ значениями n, $n-1$ соответственно. Например, диаграмма (37) распадается на

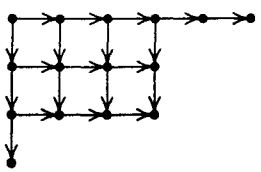
$$\begin{array}{|c|c|c|c|}\hline 1 & 2 & 3 & 6 \\ \hline 4 & 5 \\ \cline{1-2}\end{array} \quad \text{и} \quad \begin{array}{|c|c|c|c|}\cline{3-4} \multicolumn{2}{c|}{} & 7 & 10 \\ \hline 8 & 9 & 11 & 12 \\ \hline\end{array}\,;$$

после поворота и перенумерации имеем

$$P = \begin{array}{|c|c|c|c|}\hline 1 & 2 & 3 & 6 \\ \hline 4 & 5 \\ \cline{1-2}\end{array}\,, \quad Q = \begin{array}{|c|c|c|c|}\hline 1 & 2 & 4 & 5 \\ \hline 3 & 6 \\ \cline{1-2}\end{array}\,. \tag{38}$$

Наоборот, каждой паре диаграмм одинаковой формы, состоящих из n элементов и из не более чем двух строк, соответствует диаграмма формы (n, n). Следовательно (из упр. 7), *число перестановок $a_1\, a_2 \ldots a_n$ множества $\{1, 2, \ldots, n\}$, не содержащих убывающих подпоследовательностей $a_i > a_j > a_k$ при $i < j < k$, равно числу бинарных деревьев с n узлами.* Интересное взаимно однозначное соответствие между такими перестановками и бинарными деревьями, установленное более прямым способом, чем тот окольный путь через алгоритм I, которым воспользовались мы, нашел Д. Ротем [см. D. Rotem, *Inf. Proc. Letters* **4** (1975), 58–61]; аналогично существует довольно прямое соответствие между бинарными деревьями и перестановками, не содержащими подпоследовательностей $a_j > a_k > a_i$ при $i < j < k$ (см. упр. 2.2.1–5).

Число способов заполнения диаграммы формы (6,4,4,1), очевидно, равно числу способов разметки вершин ориентированного графа



(39)

числами $\{1, 2, \ldots, 15\}$ так, чтобы метка вершины u была меньше метки вершины v, если $u \to v$. Иначе говоря, оно равно числу способов топологической сортировки частичного упорядочения (39), как в разделе 2.2.3.

В общем случае можно сформулировать эту же задачу для произвольного ориентированного графа, не содержащего ориентированных циклов. Было бы хорошо, если бы существовала какая-нибудь простая формула, обобщающая теорему H для произвольного графа, однако не все графы обладают такими приятными свойствами, как графы, соответствующие диаграммам. Другие классы ориентированных графов, для которых задача о разметке вершин имеет простое решение, частично обсуждаются в упражнениях в конце этого раздела. Там также приводятся упражнения, в которых показано, что для некоторых вариантов ориентированных графов *не* существует простой формулы, соответствующей теореме H. Например, число способов разметки вершин не всегда является делителем $n!$.

Завершим наши исследования подсчетом общего числа диаграмм, которые можно сформировать из n различных элементов. Обозначим это число через t_n. Как вытекает из следствия теоремы B, t_n равно числу инволюций множества $\{1, 2, \ldots, n\}$. Перестановка обратна самой себе тогда и только тогда, когда ее представление с помощью циклов состоит только из единичных циклов (неподвижных точек) и циклов из двух элементов (транспозиций). Поскольку в t_{n-1} из t_n инволюций (n) — неподвижная точка, а в t_{n-2} из них $(j\ n)$ — цикл из двух элементов при некотором фиксированном $j < n$, то получаем формулу

$$t_n = t_{n-1} + (n-1)t_{n-2}\,, \tag{40}$$

которую вывел Роте в 1800 годудля получения таблицы значений t_n при малых n. Значения для $n \geq 0$ равны 1, 1, 2, 4, 10, 26, 76, 232, 764, 2 620, 9 496, ...

Рассмотрим другой способ. Пусть имеется k циклов из двух элементов и $(n-2k)$ единичных циклов. Существует $\binom{n}{2k}$ способов выбора неподвижных точек, и полиномиальный коэффициент $(2k)!/(2!)^k$ равен числу способов упорядочения остальных элементов в k различимых транспозиций. Разделив на $k!$, чтобы сделать эти транспозиции неразличимыми, получим

$$t_n = \sum_{k=0}^{\lfloor n/2 \rfloor} t_n(k), \qquad t_n(k) = \frac{n!}{(n-2k)!\,2^k k!}\,. \tag{41}$$

К сожалению, насколько это известно, такую сумму уже нельзя упростить (если только не принимать во внимание использование полиномов Эрмита $i^n 2^{-n/2} \times$ $\times H_n\left(-i/\sqrt{2}\right)$ в качестве отсчетов). Поэтому, для того чтобы лучше понять поведение величины t_n, мы применим два косвенных подхода.

a) Можно найти производящую функцию

$$\sum_n t_n z^n/n! = e^{z+z^2/2} \tag{42}$$

(см. упр. 25).

b) Можно определить асимптотическое поведение величины t_n. Это поучительная задача, и ее решение содержит несколько общих методов, которые будут полез-

ны и в связи с другими вопросами; поэтому конец раздела мы посвятим анализу асимптотического поведения t_n.

Первый шаг в анализе асимптотического поведения (41) состоит в выделении слагаемых, делающих основной вклад в сумму. Поскольку

$$\frac{t_n(k+1)}{t_n(k)} = \frac{(n-2k)(n-2k-1)}{2(k+1)}, \tag{43}$$

можно видеть, что слагаемые постепенно возрастают от $k = 0$ до $t_n(k+1) \approx t_n(k)$, когда k приблизительно равно $\frac{1}{2}(n-\sqrt{n})$, а затем убывают до нуля, когда k достигает значения, равного примерно $\frac{1}{2}n$. Ясно, что основной вклад вносят члены в окрестности значения $k = \frac{1}{2}(n-\sqrt{n})$. Для анализа, однако, удобнее, чтобы основной вклад достигался при значении 0, поэтому представим k в виде

$$k = \tfrac{1}{2}(n-\sqrt{n}) + x \tag{44}$$

и исследуем величину $t_n(k)$ как функцию от x.

Чтобы избавиться от факториалов в $t_n(k)$, полезно воспользоваться формулой Стирлинга 1.2.11.2–(18). Для этой цели удобно (как мы вскоре увидим) ограничить область изменения x промежутком

$$-n^{\epsilon+1/4} \le x \le n^{\epsilon+1/4}, \tag{45}$$

где ϵ равно, скажем, 0.001, так что можно включить слагаемое погрешности. После довольно трудоемких вычислений, которые автор в середине 60-х годов выполнил ручкой на бумаге и которые теперь могут быть реализованы средствами компьютерной алгебры, получается формула

$$\begin{aligned} t_n(k) = \exp\bigl(&\tfrac{1}{2}n\ln n - \tfrac{1}{2}n + \sqrt{n} - \tfrac{1}{4}\ln n - 2x^2/\sqrt{n} - \tfrac{1}{4} - \tfrac{1}{2}\ln\pi \\ &- \tfrac{4}{3}x^3/n + 2x/\sqrt{n} + \tfrac{1}{3}/\sqrt{n} - \tfrac{4}{3}x^4/n\sqrt{n} + O(n^{5\epsilon-3/4})\bigr). \end{aligned} \tag{46}$$

Ограничение (45) на x оправдывается тем, что можно положить $x = \pm n^{\epsilon+1/4}$ для получения верхней оценки всех отброшенных слагаемых, а именно

$$e^{-2n^{2\epsilon}} \exp\bigl(\tfrac{1}{2}n\ln n - \tfrac{1}{2}n + \sqrt{n} - \tfrac{1}{4}\ln n - \tfrac{1}{4} - \tfrac{1}{2}\ln\pi + O(n^{3\epsilon-1/4})\bigr). \tag{47}$$

Если же умножить это на n, получится верхняя оценка суммы исключенных слагаемых. Верхняя оценка имеет меньший порядок, чем те слагаемые, которые вычислим для x в ограниченном промежутке (45), благодаря множителю $\exp(-2n^{2\epsilon})$, намного меньшему любого полинома от n.

Очевидно, можно отбросить в сумме множитель

$$\exp\bigl(\tfrac{1}{2}n\ln n - \tfrac{1}{2}n + \sqrt{n} - \tfrac{1}{4}\ln n - \tfrac{1}{4} - \tfrac{1}{2}\ln\pi + \tfrac{1}{3}/\sqrt{n}\bigr), \tag{48}$$

после чего нам останется только просуммировать

$$\begin{aligned} &\exp\bigl(-2x^2/\sqrt{n} - \tfrac{4}{3}x^3/n + 2x/\sqrt{n} - \tfrac{4}{3}x^4/n\sqrt{n} + O(n^{5\epsilon-3/4})\bigr) \\ &\quad = \exp\left(\frac{-2x^2}{\sqrt{n}}\right)\left(1 - \frac{4}{3}\frac{x^3}{n} + \frac{8}{9}\frac{x^6}{n^2}\right)\left(1 + 2\frac{x}{\sqrt{n}} + 2\frac{x^2}{n}\right) \\ &\qquad \times \left(1 - \frac{4}{3}\frac{x^4}{n\sqrt{n}}\right)\bigl(1 + O(n^{9\epsilon-3/4})\bigr) \end{aligned} \tag{49}$$

по всем $x = \alpha, \alpha+1, \ldots, \beta-2, \beta-1$, где $-\alpha$ и β (необязательно целые) равны приблизительно $n^{\epsilon+1/4}$. Если сместить интервал суммирования, то формулу суммирования Эйлера (1.2.11.2–(10)) можно записать в виде

$$\sum_{\alpha\le x<\beta} f(x) = \int_\alpha^\beta f(x)\,dx - \frac{1}{2}f(x)\Big|_\alpha^\beta$$

$$+\frac{1}{2}B_2\,\frac{f'(x)}{1!}\Big|_\alpha^\beta + \cdots + \frac{1}{m+1}\,B_{m+1}\,\frac{f^{(m)}(x)}{m!}\Big|_\alpha^\beta + R_{m+1}\,. \qquad (50)$$

Здесь $|R_m| \le (4/(2\pi)^m)\int_\alpha^\beta \big|f^{(m)}(x)\big|\,dx$. Если положить $f(x) = x^t \exp\big(-2x^2/\sqrt{n}\,\big)$, где t — фиксированное неотрицательное целое число, найдем из формулы суммирования Эйлера асимптотический ряд для $\sum f(x)$ при $n\to\infty$, поскольку

$$f^{(m)}(x) = n^{(t-m)/4} g^{(m)}(n^{-1/4}x), \qquad g(y) = y^t e^{-2y^2}\ . \qquad (51)$$

и $g(y)$ — хорошая функция, не зависящая от n. Производная $g^{(m)}(y)$ равна e^{-2y^2}, умноженному на полином от y. Следовательно, $R_m = O(n^{(t+1-m)/4})\int_{-\infty}^{+\infty}|g^{(m)}(y)|\,dy = O(n^{(t+1-m)/4})$. Далее, поменяв α и β на $-\infty$ и $+\infty$ в правой части уравнения (50), мы сведем ошибку к величине, не превышающей $O\big(\exp(-2n^\epsilon)\big)$ в каждом члене. Таким образом,

$$\sum_{\alpha\le x<\beta} f(x) = \int_{-\infty}^{\infty} f(x)\,dx + O(n^{-m}) \qquad \text{для всех } m\ge 0. \qquad (52)$$

На самом деле при этом конкретном выборе $f(x)$ существен только интеграл! Интеграл нетрудно вычислить (см. упр. 26), значит, мы можем выполнить умножение и сложение в формуле (49) и получить $\sqrt{\pi/2}\big(n^{1/4} - \frac{1}{24}n^{-1/4} + O(n^{-1/2})\big)$. Таким образом,

$$t_n = \frac{1}{\sqrt{2}}n^{n/2}e^{-n/2+\sqrt{n}-1/4}\big(1 + \tfrac{7}{24}n^{-1/2} + O(n^{-3/4})\big). \qquad (53)$$

В действительности члены с O должны еще содержать дополнительно 9ϵ в экспоненте, но из наших выкладок ясно, что величина 9ϵ должна исчезнуть, если произвести вычисления с большей точностью. В принципе, этот метод можно расширить и вместо $O(n^{-3/4})$ получить $O(n^{-k})$ при любом k. Этот асимптотический ряд для t_n впервые нашли (другим способом) Мозер (Moser) и Уаймэн (Wyman) [*Canadian J. Math.* **7** (1955), 159–168].

Метод, использованный при выводе соотношения (53), представляет собой исключительно полезную методику асимптотического анализа, которая была предложена П. С. Лапласом [см. P. S. Laplace, *Mémoires Acad. Sci. Paris* (1782), 1–88]; она также анализируется в *CMath* §9.4 под наименованием "trading tails". Другие примеры и развитие метода, примененного для вывода формулы (53), приводятся в конце раздела 5.2.2.

УПРАЖНЕНИЯ

1. [*16*] Какие диаграммы (P, Q) соответствуют двухстрочному массиву

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 6 & 4 & 9 & 5 & 7 & 1 & 2 & 8 & 3 \end{pmatrix}$$

в построении из теоремы А? Какой двухстрочный массив соответствует паре диаграмм

$$P = \begin{array}{|c|c|c|}\hline 1 & 4 & 7 \\ \hline 2 & 8 \\ \cline{1-2} 5 & 9 \\ \cline{1-2}\end{array}\ , \quad Q = \begin{array}{|c|c|c|}\hline 1 & 3 & 7 \\ \hline 4 & 5 \\ \cline{1-2} 8 & 9 \\ \cline{1-2}\end{array}\ ?$$

2. [*M21*] Докажите, что (q, p) принадлежит классу t относительно массива (16) тогда и только тогда, когда t равно максимальному числу индексов $i_1, \ldots, i_t$, таких, что

$$p_{i_1} < p_{i_2} < \cdots < p_{i_t} = p, \qquad q_{i_1} < q_{i_2} < \cdots < q_{i_t} = q.$$

▶ **3.** [*M24*] Покажите, что соответствие, определенное в доказательстве теоремы А, можно также установить, построив следующую таблицу:

Строка 0	1	3	5	6	8
Строка 1	7	2	9	5	3
Строка 2	∞	7	∞	9	5
Строка 3		∞		∞	7
Строка 4					∞

Здесь в строках 0 и 1 записан заданный двухстрочный массив. При $k \geq 1$ строка $k+1$ образуется из строки k следующим образом.

a) Присвоить $p \leftarrow \infty$.
b) Пусть столбец j — крайний слева столбец, в строке k которого содержится целое число $< p$, а соответствующее место в строке $k+1$ не заполнено. Если такого столбца нет и $p = \infty$, то формирование строки $k+1$ на этом завершается; если такого столбца нет и $p < \infty$, то возвратиться к (a).
c) Вставить p в столбец j в строке $k+1$, присвоить p значение элемента столбца j и строки k и вернуться к (b).

После того как таблица таким образом построена, строка k диаграммы P составляется из тех целых чисел строки k этой таблицы, которые отсутствуют в строке $(k+1)$; строка k диаграммы Q строится из тех целых чисел строки 0, которые находятся в столбцах, содержащих ∞ в строке $k+1$.

▶ **4.** [*M30*] Пусть $a_1 \ldots a_{j-1}\, a_j \ldots a_n$ — перестановка различных элементов и $1 < j \leq n$. Перестановка $a_1 \ldots a_{j-2}\, a_j\, a_{j-1}\, a_{j+1} \ldots a_n$, получающаяся из исходной, если поменять местами a_{j-1} и a_j, называется допустимой, если либо

i) $j \geq 3$ и a_{j-2} лежит между a_{j-1} и a_j, либо
ii) $j < n$ и a_{j+1} лежит между a_{j-1} и a_j.

Например, в перестановке 1 5 4 6 8 3 7 можно выполнить ровно три допустимые замены: можно поменять местами 1 и 5, поскольку $1 < 4 < 5$; можно поменять местами 8 и 3, поскольку $3 < 6 < 8$ (или $3 < 7 < 8$); однако нельзя менять местами 5 и 4 или 3 и 7.

a) Докажите, что при допустимой замене не меняется диаграмма P, которая получается из перестановки путем последовательной вставки элементов $a_1, a_2, \ldots, a_n$ в первоначально пустую диаграмму.
b) Обратно, покажите, что можно преобразовать одну в другую любые две перестановки, которые соответствуют одной и той же диаграмме P, выполнив одну или более допустимых замен. [*Указание.* Покажите, что если диаграмма P имеет форму $(n_1, n_2, \ldots, n_m)$, то любую соответствующую ей перестановку при помощи последовательности допустимых замен можно преобразовать в "каноническую перестановку" $P_{m1} \ldots P_{mn_m} \ldots P_{21} \ldots P_{2n_2}\, P_{11} \ldots P_{1n_1}$.]

▶ **5.** [*M22*] Пусть P — диаграмма, соответствующая перестановке $a_1\,a_2\ldots a_n$. Используя результаты упр. 4, докажите, что диаграмма P^T соответствует перестановке $6a_n\ldots a_2\,a_1$.

6. [*M26*] (М. П. Шуценберже (M. P. Schützenberger).) Пусть π — инволюция с k неподвижными точками. Покажите, что диаграмма, соответствующая π в доказательстве следствия из теоремы B, содержит ровно k столбцов нечетной длины.

7. [*M20*] (К. Шенстед (C. Schensted).) Пусть P — диаграмма, соответствующая перестановке $a_1\,a_2\ldots\,a_n$. Докажите, что число *столбцов* в P равно длине j максимальной возрастающей подпоследовательности $a_{i_1} < a_{i_2} < \cdots < a_{i_j}$, где $i_1 < i_2 < \cdots < i_j$; число *строк* в P равно длине r максимальной убывающей подпоследовательности $a_{j_1} > a_{j_2} > \cdots > a_{j_r}$, где $j_1 < j_2 < \cdots < j_r$.

8. [*M18*] (П. Эрдеш (P. Erdős), Г. Секереш (G. Szekeres).) Докажите, что в любой перестановке, состоящей из более чем n^2 элементов, имеется монотонная подпоследовательность длиной более n; однако существуют перестановки n^2 элементов, не содержащие монотонных подпоследовательностей длиной более n. [*Указание.* См. предыдущее упражнение.]

9. [*M24*] Продолжая упр. 8, найдите "простую" формулу точного числа перестановок множества $\{1, 2, \ldots, n^2\}$, не содержащих монотонных подпоследовательностей длиной более n.

10. [*M20*] Докажите, что массив P остается диаграммой после окончания выполнения алгоритма S, если он был диаграммой до этого.

11. [*20*] Можно ли восстановить исходный вид диаграммы P по окончании выполнения алгоритма S, если известны только значения r и s?

12. [*M24*] Сколько раз выполняется шаг S3, если алгоритм S многократно применяется для исключения всех элементов из диаграммы P формы $(n_1, n_2, \ldots, n_m)$? Каково минимальное значение этой величины по всем формам, таким, что $n_1 + n_2 + \cdots + n_m = n$?

13. [*M28*] Докажите теорему C.

14. [*M43*] Найдите более прямое доказательство теоремы D, п. (c).

15. [*M20*] Сколько перестановок мультимножества $\{l \cdot a,\ m \cdot b,\ n \cdot c\}$ обладают следующим свойством: если читать перестановку слева направо, то число прочитанных букв c никогда не превышает числа букв b, которое, в свою очередь, не превышает числа букв a? (Например, перестановка $a\ a\ b\ c\ a\ b\ b\ c\ a\ c\ a$ обладает этим свойством.)

16. [*M08*] Сколькими способами можно топологически рассортировать частичное упорядочение, представляемое графом (39)?

17. [*HM25*] Пусть

$$g(x_1, x_2, \ldots, x_n;\, y) = x_1\,\Delta(x_1{+}y, x_2, \ldots, x_n) + x_2\,\Delta(x_1, x_2{+}y, \ldots, x_n) + \cdots + x_n\,\Delta(x_1, x_2, \ldots, x_n{+}y).$$

Докажите, что

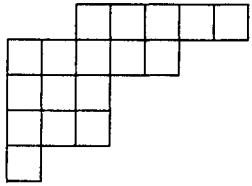
$$g(x_1, x_2, \ldots, x_n;\, y) = \left(x_1 + x_2 + \cdots + x_n + \tbinom{n}{2}\,y\right)\,\Delta(x_1, x_2, \ldots, x_n).$$

[*Указание.* Полином g однородный (все слагаемые имеют одинаковую суммарную степень) и антисимметричный по x (знак g изменится, если поменять местами x_i и x_j).]

18. [*HM30*] Обобщая упр. 17, вычислите при $m \geq 0$ сумму

$$x_1^m\,\Delta(x_1{+}y, x_2, \ldots, x_n) + x_2^m\,\Delta(x_1, x_2{+}y, \ldots, x_n) + \cdots + x_n^m\,\Delta(x_1, x_2, \ldots, x_n{+}y).$$

19. [*M40*] Найдите формулу для определения числа способов, которыми можно заполнить массив, подобный диаграмме, но без первых двух клеток в первой строке, например массив такой формы:

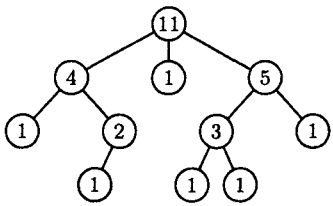


n_1-2 клеток
n_2 клеток
n_3 клеток .
$\vdots$

(Элементы в строках и столбцах должны располагаться в порядке возрастания, как в обычных диаграммах.)

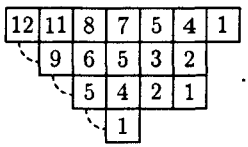
Иными словами, сколько диаграмм формы $(n_1, n_2, \ldots, n_m)$, составленных из элементов $\{1, 2, \ldots, n_1+\cdots+n_m\}$, содержат элементы 1 и 2 в первой строке?

▸ **20.** [*M24*] Докажите, что число способов такой разметки узлов данного бинарного дерева элементами $\{1, 2, \ldots, n\}$, чтобы метка каждого узла была меньше метки любого из его потомков, равно частному от деления $n!$ на произведение "длин поддеревьев", т. е. количеств узлов в каждом поддереве (см. теорему H). Например, число способов разметки узлов дерева



равно $11!\,/\,11\cdot 4\cdot 1\cdot 5\cdot 1\cdot 2\cdot 3\cdot 1\cdot 1$ ło$t1\cdot 1 = 10\cdot 9\cdot 8\cdot 7\cdot 6$ (ср. с теоремой H).

21. [*HM31*] (Р. М. Тролл (R. M. Thrall).) Пусть числа $n_1 > n_2 > \cdots > n_m$ описывают форму "сдвинутой диаграммы", в которой строка $i+1$ начинается на одну позицию правее, чем строка i; например, сдвинутая диаграмма формы $(7, 5, 4, 1)$ имеет вид



.

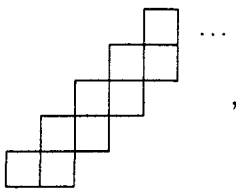
Докажите, что число способов такого заполнения сдвинутой диаграммы формы $(n_1, n_2, \ldots, n_m)$ числами $1, 2, \ldots, n = n_1+n_2+\cdots+n_m$, чтобы числа во всех строках и столбцах располагались в порядке возрастания, равно частному от деления $n!$ на произведение "длин обобщенных уголков"; на рисунке заштрихован обобщенный уголок длиной 11, соответствующий клетке строки 1 и столбца 2. (Уголки в левой части массива, имеющей вид перевернутой лестницы, имеют форму буквы U, повернутой на 90°, а не буквы L.) Итак, существует

$$17!/12\cdot 11\cdot 8\cdot 7\cdot 5\cdot 4\cdot 1\cdot 9\cdot 6\cdot 5\cdot 3\cdot 2\cdot 5\cdot 4\cdot 2\cdot 1\cdot 1$$

способов такого заполнения изображенной выше формы, чтобы элементы во всех строках и столбцах располагались в порядке возрастания.

22. [*M39*] Сколькими способами можно заполнить массив формы $(n_1, n_2, \ldots, n_m)$ элементами множества $\{1, 2, \ldots, N\}$, *если допускаются одинаковые элементы*, причем в строках элементы должны располагаться в порядке неубывания, а в столбцах — только в порядке возрастания? Например, простую форму из m строк $(1, 1, \ldots, 1)$ можно заполнить $\binom{N}{m}$ способами; форму из одной строки (m) можно заполнить $\binom{m+N-1}{m}$ способами; форму $(2, 2)$ можно заполнить $\frac{1}{3}\binom{N+1}{2}\binom{N}{2}$ способами.

▶ **23.** [*HM30*] (Д. Андрэ (D. André).) Чему равно число способов (A_n) такого заполнения числами $\{1, 2, \ldots, n\}$ массива из n ячеек



чтобы во всех строках и столбцах они располагались в порядке возрастания? Найдите производящую функцию $g(z) = \sum A_n z^n/n!$.

24. [*M28*] Докажите, что

$$\sum_{\substack{q_1+\cdots+q_n=t \\ 0\le q_1,\ldots,q_n\le m}} \binom{m}{q_1}\cdots\binom{m}{q_n}\Delta(q_1,\ldots,q_n)^2$$

$$= n!\binom{nm-(n^2-n)}{t-\frac{1}{2}(n^2-n)}\binom{m}{n-1}\binom{m}{n-2}\cdots\binom{m}{0}\Delta(n-1,\ldots,0)^2.$$

[*Указания.* Докажите, что $\Delta(k_1+n-1,\ldots,k_n) = \Delta(m-k_n+n-1,\ldots,m-k_1)$; разложите диаграмму формы $n\times(m-n+1)$ способом, аналогичным (38); преобразуйте сумму, как при выводе тождества (36).]

25. [*M20*] Почему (42) является производящей функцией для инволюций?

26. [*HM21*] Вычислите $\int_{-\infty}^{\infty} x^t \exp(-2x^2/\sqrt{n})\,dx$ при неотрицательном t.

27. [*M24*] Пусть Q — диаграмма Юнга из элементов $\{1, 2, \ldots, n\}$ и пусть элемент i находится в строке r_i и столбце c_i. Мы говорим, что i "выше" j, если $r_i < r_j$.

a) Докажите, что при $1 \le i < n$ элемент i выше $i+1$ тогда и только тогда, когда $c_i \ge c_{i+1}$.

b) Пусть Q такое, что (P, Q) соответствуют перестановке

$$\begin{pmatrix} 1 & 2 & \ldots & n \\ a_1 & a_2 & \ldots & a_n \end{pmatrix}.$$

Докажите, что i выше $i+1$ тогда и только тогда, когда $a_i > a_{i+1}$. (Следовательно, можно найти число отрезков перестановки, зная только Q. Этот результат получен Шуценберже.)

c) Докажите, что при $1 \le i < n$ элемент i выше $i+1$ в Q тогда и только тогда, когда $i+1$ выше i в Q^S.

28. [*M43*] Докажите, что средняя длина самой длинной возрастающей подпоследовательности в случайной перестановке множества $\{1, 2, \ldots, n\}$ асимптотически приближается к $2\sqrt{n}$? (Это средняя длина первой строки в соответствии из теоремы A.

29. [*HM25*] Докажите, что случайная перестановка n элементов имеет возрастающую подпоследовательность длиной $\ge l$ с вероятностью $\le \binom{n}{l}/l!$. Эта вероятность равна $O(1/\sqrt{n})$, если $l = e\sqrt{n} + O(1)$, и $O(\exp(-c\sqrt{n}))$, если $l = 3\sqrt{n}$, $c = 6\ln 3 - 6$.

30. [*M41*] (М. П. Шуценберже (M. P. Schützenberger).) Покажите, что операция перехода от P к P^S — частный случай операции, которую можно связать с *любым* конечным частично упорядоченным множеством, а не только с диаграммой. Пометьте элементы частично упорядоченного множества целыми числами $\{1, 2, \ldots, n\}$ так, чтобы эта система меток была согласована с частичным упорядочением. Найдите двойственную систему меток, аналогичную (26), путем последовательного удаления меток 1, 2, ..., передвигая

при этом другие метки так, как в алгоритме S, и помещая метки *1*, *2*, ... на освободившиеся места. Покажите, что эта операция, если ее многократно применять к двойственной системе меток с обратным отношением порядка для чисел, дает исходную систему меток; исследуйте другие свойства этой операции.

31. [*HM30*] Пусть x_n — число способов такого размещения n взаимно неатакующих ладей на шахматной доске размером $n \times n$, что расположение не меняется при отражении доски относительно обеих главных диагоналей. В результате получим $x_4 = 6$. (От инволюций же требуется симметрия только относительно одной из главных диагоналей. Похожая задача рассматривалась в упр. 5.1.3–19.) Найдите асимптотическое поведение x_n.

32. [*HM21*] Докажите, что t_n — среднее значение X^n, если X является нормальной случайной величиной с математическим ожиданием 1 и дисперсией 1.

33. [*M25*] (О. Митчелл (O. Mitchell), 1881.) Верно ли утверждение:

$$\Delta(a_1, a_2, \ldots, a_m)/\Delta(1, 2, \ldots, m)$$

является целым числом, если a_1, a_2, ..., a_m — также целые числа.

34. [*25*] (Т. Накаяма (T. Nakayama), 1940.) Докажите, что если форма диаграммы включает уголок длиной ab, она содержит и уголок длиной a.

▸ **35.** [*30*] (А. П. Хиллман (A. P. Hillman) и Р. М. Грассл (R. M. Grassl), 1976.) Размещение неотрицательных целых чисел p_{ij} по ячейкам диаграммы, которая имеет n_i ячеек в строке i и n'_j ячеек в столбце j, называется *плоским разбиением* m, если $\sum p_{ij} = m$ и

$$p_{i1} \geq \cdots \geq p_{in_i}, \qquad p_{1j} \geq \cdots \geq p_{n'_j j}, \qquad \text{где } 1 \leq i \leq n'_1,\ 1 \leq j \leq n_1.$$

Оно же называется *обратным плоским разбиением*, если вместо сформулированных выше выполняются условия

$$p_{i1} \leq \cdots \leq p_{in_i}, \qquad p_{1j} \leq \cdots \leq p_{n'_j j}, \qquad \text{где } 1 \leq i \leq n'_1,\ 1 \leq j \leq n_1.$$

Рассмотрите следующий алгоритм, который выполняет обратные плоские разбиения данной формы и формирует другой массив чисел q_{ij}, имеющий ту же форму.

G1. [Инициализация.] Присвоить $q_{ij} \leftarrow 0$, $1 \leq j \leq n_i$ и $1 \leq i \leq n'_1$. Затем присвоить $j \leftarrow 1$.

G2. [Поиск ненулевой ячейки.] Если $p_{n'_j j} > 0$, присвоить $i \leftarrow n'_j$, $k \leftarrow j$ и перейти к шагу G3. В противном случае, если $j < n_1$, увеличить j на 1 и повторить этот шаг. В противном случае остановить процесс (массив p теперь обнулен).

G3. [Уменьшение p.] Уменьшить p_{ik} на 1.

G4. [Переход вверх или вправо.] Если $i > 1$ и $p_{(i-1)k} > p_{ik}$, уменьшить i на 1 и вернуться к шагу G3. В противном случае, если $k < n_i$, увеличить k на 1 и вернуться к шагу G3.

G5. [Увеличение q.] Увеличить q_{ij} на 1 и вернуться к шагу G2. ▌

Разработайте алгоритм, который восстанавливает значения p по значениям q, и тем самым докажите, что описанное выше построение определяет взаимно однозначное соответствие между обратным плоским разбиением m и решением уравнения

$$m = \sum h_{ij} q_{ij}\,,$$

где числа h_{ij} — длины уголков формы.

36. [*HM27*] (Р. П. Стэнли (R. P. Stanley), 1971.) (a) Докажите, что количество обратных плоских разбиений m на данной форме равно $[z^m]\,1/\prod(1-z^{h_{ij}})$, где числа h_{ij} — суть длины уголков формы. (b) Выведите из этого результата теорему H. [*Указание.* Задумайтесь, к чему асимптотически приближается количество разбиений при $m \to \infty$.]

37. [*M20*] (П. А. Мак-Магон , 1912.) Какова производящая функция для любого плоского разбиения? (Коэффициент при z^m должен быть общим числом разбиений m, если форма диаграммы неограниченна.)

▸ **38.** [*M30*] (Грин (Greene), Ниенхьюз (Nijenhuis), Вильф (Wilf), 1979.) Можно построить прямой ациклический граф на ячейках T любой данной формы диаграммы следующим образом. Пусть дуги исходят из каждой ячейки к другим ячейкам в этом уголке; внешняя степень ячейки (i,j) будет тогда $d_{ij} = h_{ij} - 1$, где h_{ij} равно длине уголка. Предположим, мы построили некоторый случайный путь на этом диграфе, выбрав случайным образом начальную ячейку (i,j), и случайно выбирали следующие дуги до тех пор, пока не оказались в угловой ячейке, из которой нет выхода. Каждый случайный выбор выполняется равновероятно.

a) Пусть (a,b) — угловая ячейка T и пусть $I = \{i_0,\ldots,i_k\}$ и $J = \{j_0,\ldots,j_l\}$ — множества строк и столбцов, причем $i_0 < \cdots < i_k = a$ и $j_0 < \cdots < j_l = b$. Диграф содержит $\binom{k+l}{k}$ путей, множества строк и столбцов которых суть I и J; пусть $P(I,J)$ — вероятность выбора определенного случайного пути. Докажите, что $P(I,J) = 1/(n\,d_{i_0b}\ldots d_{i_{k-1}b}\,d_{aj_0}\ldots d_{aj_{l-1}})$, где $n = |T|$.

b) Пусть $f(T) = n!/\prod h_{ij}$. Докажите, что случайный путь завершается в угловой ячейке (a,b) с вероятностью $f(T\setminus\{(a,b)\})/f(T)$.

c) Покажите, что результат (b) доказывает теорему H и предоставляет способ формирования случайной диаграммы формы T, причем все диаграммы $f(T)$ равновероятны.

39. [*M38*] (И. М. Пак, А. В. Стояновский, 1992.) Пусть P — массив $(n_1,\ldots,n_m)$, который заполнен некоторой перестановкой множества целых чисел $\{1,\ldots,n\}$, $n = n_1 + \cdots + n_m$. Следующая процедура, которая аналогична алгоритму "просеивания", описанному в разделе 5.2.3, может быть использована для преобразования P в диаграмму. Она также определяет массив Q такой же формы, который может послужить для комбинаторного доказательства теоремы H.

P1. [Цикл по (i,j).] Выполнить шаги P2 и P3 для каждой ячейки (i,j) массива в обратном лексикографическом порядке (т. е. снизу вверх и справа налево в каждой строке), затем прекратить выполнение процедуры.

P2. [Обработать P в (i,j).] Присвоить $K \leftarrow P_{ij}$ и выполнить алгоритм S′ (см. ниже).

P3. [Изменение Q.] Присвоить $Q_{ik} \leftarrow Q_{i(k+1)}+1$ для $j \le k < s$ и присвоить $Q_{is} \leftarrow i-r$. ▮

Ниже описан тот же алгоритм, что и алгоритм S Шуценберже, но шаги S1 и S2 слегка модифицированы — распространены на более общий случай.

S1′. [Инициализация.] Присвоить $r \leftarrow i$, $s \leftarrow j$.

S2′. [Выполнено?] Если $K \lesssim P_{(r+1)s}$ и $K \lesssim P_{r(s+1)}$, присвоить $P_{rs} \leftarrow K$ и завершить процесс.

(Алгоритм S предполагает особый случай: $i = 1$, $j = 1$, $K = \infty$.)

Например, алгоритм P обрабатывает массив формы $(3,3,2)$ следующим образом, если просматривать содержимое массивов P и Q в начале каждого шага P2, причем значения P_{ij} выделены полужирным шрифтом:

$$P = \begin{array}{|c|c|c|}\hline 7&8&5\\\hline 1&6&4\\\hline 3&\mathbf{2}\\\cline{1-2}\end{array}\quad \begin{array}{|c|c|c|}\hline 7&8&5\\\hline 1&6&4\\\hline \mathbf{3}&2\\\cline{1-2}\end{array}\quad \begin{array}{|c|c|c|}\hline 7&8&5\\\hline 1&6&\mathbf{4}\\\hline 2&3\\\cline{1-2}\end{array}\quad \begin{array}{|c|c|c|}\hline 7&8&5\\\hline 1&\mathbf{6}&4\\\hline 2&3\\\cline{1-2}\end{array}\quad \begin{array}{|c|c|c|}\hline 7&8&5\\\hline \mathbf{1}&3&4\\\hline 2&6\\\cline{1-2}\end{array}\quad \begin{array}{|c|c|c|}\hline 7&8&5\\\hline 1&3&4\\\hline 2&6\\\cline{1-2}\end{array}\quad \begin{array}{|c|c|c|}\hline 7&\mathbf{8}&4\\\hline 1&3&5\\\hline 2&6\\\cline{1-2}\end{array}\quad \begin{array}{|c|c|c|}\hline \mathbf{7}&3&4\\\hline 1&5&8\\\hline 2&6\\\cline{1-2}\end{array}$$

$$Q = \begin{array}{|c|c|c|}\hline &&\\\hline &&\\\hline &&\\\cline{1-2}\end{array}\quad \begin{array}{|c|c|c|}\hline &&\\\hline &&\\\hline &0&\\\cline{1-2}\end{array}\quad \begin{array}{|c|c|c|}\hline &&\\\hline &&\\\hline 1&0&\\\cline{1-2}\end{array}\quad \begin{array}{|c|c|c|}\hline &&\\\hline &&0\\\hline 1&0&\\\cline{1-2}\end{array}\quad \begin{array}{|c|c|c|}\hline &&\\\hline &-1&0\\\hline 1&0&\\\cline{1-2}\end{array}\quad \begin{array}{|c|c|c|}\hline &&\\\hline 0&-1&0\\\hline 1&0&\\\cline{1-2}\end{array}\quad \begin{array}{|c|c|c|}\hline &&-1\\\hline 0&-1&0\\\hline 1&0&\\\cline{1-2}\end{array}\quad \begin{array}{|c|c|c|}\hline &0&-1\\\hline 0&-1&0\\\hline 1&0&\\\cline{1-2}\end{array}\,.$$

Окончательный результат имеет вид

$$P = \begin{array}{|c|c|c|}\hline 1&3&4\\\hline 2&5&8\\\hline 6&7\\\cline{1-2}\end{array}\,,\qquad Q = \begin{array}{|c|c|c|}\hline 1&-2&-1\\\hline 0&-1&0\\\hline 1&0\\\cline{1-2}\end{array}\,.$$

a) Если P — это просто массив $1 \times n$, алгоритм P сортирует его и превращает в

$$\begin{array}{|c|c|c|}\hline 1&\dots&n\\\hline\end{array}\,.$$

Какой вид при этом будет иметь массив Q?

b) Ответьте на тот же вопрос, но в случае, если P имеет вид $n \times 1$, а не $1 \times n$.

c) Докажите, что в общем случае получим

$$-b_{ij} \le Q_{ij} \le r_{ij},$$

где b_{ij} — число ячеек, расположенных ниже ячейки (i,j), и r_{ij} — число ячеек справа от нее. Таким образом, число возможных значений Q_{ij} в точности равно h_{ij} — размеру (i,j)-го уголка.

d) Теорема H будет доказана по построению, если мы сможем показать, что алгоритм P определяет однозначное соответствие между $n!$ способами заполнения исходной формы и парами результирующих массивов (P,Q), где P есть диаграмма, а элементы в Q удовлетворяют условиям п. (c). Значит, желательно найти процедуру, обратную алгоритму P. Для каких исходных перестановок алгоритм P сформирует массив $Q = \left(\begin{smallmatrix}0&-1\\0&0\end{smallmatrix}\right)$ формы 2×2?

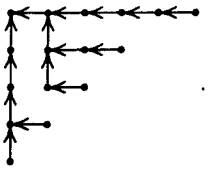
e) Какую исходную перестановку алгоритм P преобразует в массивы

$$P = \begin{array}{|c|c|c|c|c|c|}\hline 1&3&5&7&11&15\\\hline 2&6&8&14\\\cline{1-4}4&9&13\\\cline{1-3}10&12\\\cline{1-2}16\\\cline{1-1}\end{array}\,,\qquad Q = \begin{array}{|c|c|c|c|c|c|}\hline -2&-3&-1&-1&1&0\\\hline 3&-2&-1&0\\\cline{1-4}0&-1&0\\\cline{1-3}-1&0\\\cline{1-2}0\\\cline{1-1}\end{array}\,?$$

f) Разработайте алгоритм, обратный по отношению к алгоритму P, задавая любую пару массивов (P,Q), таких, что P — диаграмма, а Q удовлетворяет условиям п. (c). [*Указание.* Постройте ориентированное дерево, вершины которого — ячейки (i,j), а дуги —

$$(i,j) \to (i,j-1), \quad \text{если } P_{i(j-1)} > P_{(i-1)j};$$
$$(i,j) \to (i-1,j), \quad \text{если } P_{i(j-1)} < P_{(i-1)j}.$$

В примере (e) мы имеем дерево



Его составные элементы и представляют собой ключ к обращению алгоритма P.]

40. [*HM43*] Предположим, что некоторая диаграмма Юнга построена путем последовательного размещения чисел 1, 2, ..., n таким способом, что выбор очередной доступной ячейки для каждого следующего числа равновероятен. Например, с вероятностью $\frac{1}{1}\cdot\frac{1}{2}\cdot\frac{1}{2}\cdot\frac{1}{3}\cdot\frac{1}{3}\cdot\frac{1}{3}\cdot\frac{1}{4}\cdot\frac{1}{3}\cdot\frac{1}{3}\cdot\frac{1}{4}\cdot\frac{1}{4}\cdot\frac{1}{5}\cdot\frac{1}{4}\cdot\frac{1}{5}\cdot\frac{1}{4}$ так будет получена диаграмма (1).

Докажите, что с большой вероятностью результирующая форма $(n_1, n_2, \ldots, n_m)$ будет иметь $m \approx \sqrt{6n}$ и $\sqrt{k} + \sqrt{n_{k+1}} \approx \sqrt{m}$ при $0 \le k \le m$.

41. [*25*] (*Беспорядок в библиотеке.*) Некоторые бестолковые читатели часто возвращают книги в библиотеку и ставят их на полки где попало. Измерить вносимый при этом беспорядок можно, рассмотрев минимальное количество повторений простейшей процедуры перемещения данной книги, которое необходимо для того, чтобы все книги заняли надлежащие им места.

Пусть $\pi = a_1 a_2 \ldots a_n$ — перестановка множества $\{1, 2, \ldots, n\}$. Операция удаления-вставки превращает π в

$$a_1 \ldots a_{i-1}\, a_{i+1} \ldots a_j\, a_i\, a_{j+1} \ldots a_n \qquad \text{или} \qquad a_1 \ldots a_j\, a_i\, a_{j+1} \ldots a_{i-1}\, a_{i+1} \ldots a_n$$

для некоторых i и j. Пусть $\mathrm{dis}(\pi)$ — минимальное число операций удаления-вставки, которое рассортирует π и приведет его в порядок. Можно ли выразить $\mathrm{dis}(\pi)$ в терминах простых характеристик π?

▶ **42.** [*30*] (*Беспорядок генов.*) Молекула ДНК растения *Lobelia fervens* имеет гены в последовательности $g_7^R g_1 g_2 g_4 g_5 g_3 g_6^R$, где g_7^R обозначает отражение слева направо g_7. Те же самые гены существуют и в табаке, но в порядке $g_1 g_2 g_3 g_4 g_5 g_6 g_7$. Покажите, что для того, чтобы получить из $g_1 g_2 g_3 g_4 g_5 g_6 g_7$ последовательность $g_7^R g_1 g_2 g_4 g_5 g_3 g_6^R$, необходимо пять операций "перебрасывания" подцепочек. (Операция перебрасывания превращает $\alpha\beta\gamma$ в $\alpha\beta^R\gamma$, если α, β и γ являются цепочками.)

43. [*35*] Продолжая предыдущее упражнение, покажите, что по крайней мере $n+1$ операций перебрасывания необходимо для сортировки любого варианта компоновки $g_1 g_2 \ldots g_n$. Постройте примеры, которые требуют $n+1$ операций для всех $n > 3$.

44. [*M37*] Покажите, что среднее число перебрасываний, необходимых для сортировки случайной компоновки n генов, больше, чем $n - H_n$, если все $2^n\, n!$ компоновок генов равновероятны.

5.2. ВНУТРЕННЯЯ СОРТИРОВКА

НАЧНЕМ обсуждение "хорошего" процесса сортировки с маленького эксперимента. Как бы вы решили следующую задачу программирования?

> "В ячейках памяти R+1, R+2, R+3, R+4 и R+5 содержится пять чисел. Напишите программу, которая переразмещает, если нужно, эти числа так, чтобы они расположились в порядке возрастания."

(Если вы уже знакомы с какими-либо методами сортировки, постарайтесь, пожалуйста, на минуту забыть о них; вообразите, что вы решаете такую задачу впервые, не имея никакого представления о том, как к ней подступиться.)

Прежде чем читать дальше, настоятельно просим вас найти хоть какое-нибудь решение этой задачи.

. . . . . .

Время, затраченное на решение приведенной выше задачи, окупится с лихвой, когда вы продолжите чтение этой главы. Возможно, ваше решение окажется одним из следующих.

A. *Сортировка методом вставок.* Элементы просматриваются по одному, и каждый новый элемент вставляется в подходящее место среди ранее упорядоченных элементов. (Именно таким способом игроки в бридж упорядочивают свои карты, беря по одной.)

B. *Обменная сортировка.* Если два элемента расположены не по порядку, то они меняются местами. Этот процесс повторяется до тех пор, пока элементы не будут упорядочены.

C. *Сортировка посредством выбора.* Сначала выделяется наименьший (или, быть может, наибольший) элемент и каким-либо образом отделяется от остальных, затем выбирается наименьший (наибольший) из оставшихся и т. д.

D. *Сортировка путем подсчета.* Каждый элемент сравнивается со всеми остальными; окончательное положение элемента определяется после подсчета числа меньших ключей.

E. *Специальная сортировка.* Она хороша для пяти элементов, указанных в задаче, но не поддается простому обобщению, если элементов больше.

F. *Способ лентяя.* Вы не откликнулись на наше предложение и отказались решать задачу. Жаль — вы упустили свой шанс... Если вы дошли до этого места, то и так уже слишком много знаете.

G. *Новая суперметодика сортировки.* Это существенно усовершенствованные известные методы. (Если вы нашли такую методику, пожалуйста, немедленно сообщите об этом автору.)

Если бы данная задача была сформулирована, скажем, для 1 000 элементов, а не для 5, то вы могли бы открыть и более тонкие методы, о которых речь пойдет ниже. В любом случае, приступая к новой задаче, разумно найти какую-нибудь очевидную процедуру, а затем попытаться улучшить ее. Развитие вариантов A, B и C приводит к появлению важных классов методов сортировки, которые представляют собой усовершенствованные сформулированные выше простые идеи.

Было изобретено множество различных алгоритмов сортировки, и в этой книге рассматривается около 25 из них. Такое пугающее количество методов на самом

деле — лишь малая толика всех алгоритмов, придуманных на сегодняшний день; многие уже устаревшие методы мы вовсе не будем рассматривать или упомянем лишь вскользь. Почему же существует так много методов сортировки? Применительно к программированию это частный случай вопроса "Почему существует так много методов x?", где x пробегает множество всех задач. Ответ если и неочевиден, то вполне понятен — каждый метод имеет свои преимущества и недостатки, поэтому он оказывается эффективнее других при некоторых конфигурациях данных и аппаратуры. К сожалению, неизвестен наилучший способ сортировки (если он вообще существует); имеется *много* наилучших методов, но только в случаях, когда известно, что сортируется, на каком компьютере и с какой целью. Говоря словами Редьярда Киплинга, "существует 9 и еще 60 способов сложить песню племени, и каждый из них в отдельности хорош". Полезно изучить характеристики каждого метода сортировки, чтобы для конкретного случая можно было сделать разумный выбор. К счастью, эта задача не столь уж громоздка, поскольку алгоритмы взаимосвязаны подчас самым причудливым способом.

В начале этой главы мы ввели основную терминологию и обозначения, которые и будем использовать при изучении сортировки. Записи

$$R_1, R_2, \ldots, R_N \tag{1}$$

должны быть рассортированы в порядке неубывания своих ключей $K_1, K_2, \ldots, K_N$, т. е., по существу, нужно найти перестановку $p(1)\,p(2)\ldots p(N)$, такую, что

$$K_{p(1)} \le K_{p(2)} \le \cdots \le K_{p(N)}. \tag{2}$$

В этом разделе рассматривается *внутренняя сортировка*, когда число записей, подлежащих сортировке, достаточно мало, так что весь процесс можно провести в оперативной памяти компьютера, обладающей высоким быстродействием.

В одних случаях может понадобиться физически переставлять записи в памяти так, чтобы их ключи были упорядочены; в других можно обойтись вспомогательной таблицей некоторого вида, определяющей перестановку. Если записи и/или ключи занимают несколько слов памяти, то часто лучше составить новую таблицу адресных ссылок, которые указывают на записи, и работать с этими ссылками, не перемещая громоздкие записи. Такой метод называется *сортировкой таблицы адресов* (рис. 6). Если ключи короткие, а сопутствующая информация в записях велика, то для повышения скорости ключи можно вынести в таблицу адресных ссылок. Этот процесс называется *сортировкой ключей*.

В других схемах сортировки используется вспомогательное поле связи, которое включается в каждую запись. Связи обрабатываются таким образом, что в результате все они оказываются объединенными в линейный список, в котором каждая связь указывает на следующую по порядку запись. Этот способ называется *сортировкой списка* (рис. 7).

После сортировки таблицы адресов или списка можно по желанию расположить записи в порядке неубывания. Сделать это можно несколькими способами, требующими дополнительной памяти для хранения всего одной записи (см. упр. 10 и 12); или же можно просто переместить записи в новую область памяти, если она способна вместить все эти записи. Последний способ обычно вдвое быстрее первого, но требует почти в два раза больше памяти. Во многих приложениях вовсе не

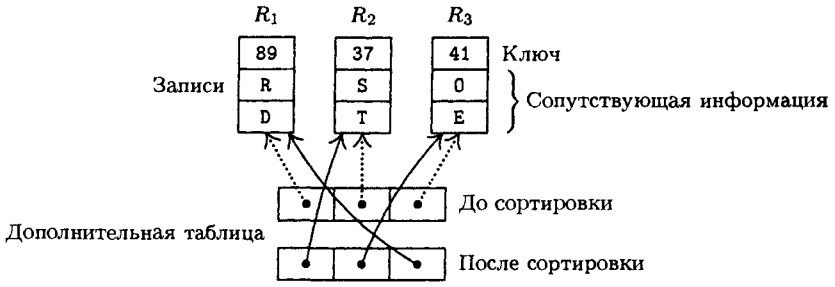


Рис. 6. Сортировка таблицы адресных ссылок.

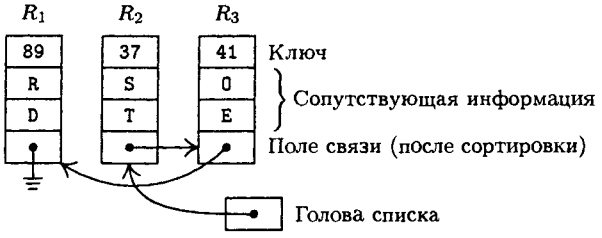


Рис. 7. Сортировка списка.

обязательно перемещать записи, так как поля связи, как правило, вполне приемлемы для всех последующих операций обработки.

Все методы сортировки, которые мы исследуем "досконально", будут проиллюстрированы четырьмя способами: посредством

a) словесного описания алгоритма,

b) блок-схемы,

c) программы для машины MIX,

d) примера применения этого метода сортировки к некоторому множеству из 16 чисел.

Программы для компьютера MIX будут, как правило, написаны в предположении, что ключ — числовой и что он помещается в одном слове; иногда мы даже будем ограничивать значения ключей так, чтобы они занимали лишь часть слова. Отношением порядка $<$ будет обычное арифметическое отношение порядка, а записи будут состоять из одного ключа, без сопутствующей информации. Эти предположения позволяют сделать программы более короткими и простыми для понимания, и не составляет труда распространить их на общий случай (например, применяя сортировку таблиц адресов). Вместе с программами для MIX приводится анализ времени выполнения соответствующего алгоритма сортировки.

Сортировка посредством подсчета. Чтобы проиллюстрировать, как мы будем анализировать методы внутренней сортировки, рассмотрим идею "подсчета", упомянутую в начале данного раздела. Этот простой метод основан на том, что j-й ключ в окончательно упорядоченной последовательности превышает ровно $j-1$ остальных ключей. Иначе говоря, если известно, что некоторый ключ превышает ровно 27 дру-

гих и если никакие два ключа не равны, то после сортировки соответствующая запись должна занять 28-е место. Таким образом, идея состоит в том, чтобы сравнить попарно все ключи и подсчитать, сколько из них меньше каждого отдельного ключа.

Очевидный способ выполнения поставленной задачи таков:

$$\big((\text{сравнить } K_j \text{ с } K_i) \text{ при } 1 \le j \le N\big) \text{ при } 1 \le i \le N;$$

но нетрудно заметить, что более половины этих операций излишни, поскольку не нужно сравнивать ключ сам с собой и после сравнения K_a с K_b уже не надо сравнивать K_b с K_a. Поэтому достаточно

$$\big((\text{сравнить } K_j \text{ с } K_i) \text{ при } 1 \le j < i\big) \text{ при } 1 < i \le N.$$

Итак, получаем следующий алгоритм.

Алгоритм C (*Подсчет сравнений*). Этот алгоритм (рис. 8) сортирует записи $R_1, \ldots, R_N$ по ключам $K_1, \ldots, K_N$, используя вспомогательную таблицу COUNT[1], ..., COUNT[N] для подсчета числа ключей, меньших данного. После завершения алгоритма величина COUNT[j] + 1 определяет окончательное положение записи R_j.

C1. [Сбросить счетчики COUNT.] Установить в счетчиках COUNT[1]–COUNT[N] нули.

C2. [Цикл по i.] Выполнить шаг C3 при $i = N, N-1, \ldots, 2$; затем завершить выполнение процедуры.

C3. [Цикл по j.] Выполнить шаг C4 при $j = i-1, i-2, \ldots, 1$.

C4. [Сравнить $K_i : K_j$.] Если $K_i < K_j$, то увеличить COUNT[j] на 1; в противном случае увеличить COUNT[i] на 1. ▌

Обратите внимание на то, что в данном алгоритме записи не перемещаются. Он аналогичен алгоритму сортировки таблицы адресов, поскольку таблица COUNT определяет конечное положение записей; но эти методы несколько различаются, потому что COUNT указывает, куда нужно переслать запись R_j, а не запись, которую надо переслать на место R_j. (Таким образом, таблица COUNT определяет перестановку, *обратную* перестановке $p(1) \ldots p(N)$; см. раздел 5.1.1.)

В табл. 1 проиллюстрирован типичный ход событий при подсчете сравнений. Использован пример с 16 случайными числами, которые были выбраны автором еще 19 марта 1963 года.Те же самые 16 чисел будут использованы в примерах, иллюстрирующих и другие методы, анализ которых еще впереди.

В рассуждении, предшествующем этому алгоритму, мы не учитывали, что ключи могут быть равными. Это, вообще говоря, серьезное упущение, которое может повлечь за собой самые опасные последствия, потому что если бы равным ключам соответствовали равные счетчики, то заключительное перемещение записей было бы довольно сложным. К счастью, как показано в упр. 2, алгоритм C дает верный результат независимо от числа равных ключей.

Программа C (*Подсчет сравнений*). Ниже приведена программа для MIX, реализующая алгоритм C в предположении, что R_j хранятся по адресу INPUT + j, а COUNT[j] — по адресу COUNT + j, где $1 \le j \le N$; состояние регистров следующее: rI1 $\equiv i$, rI2 $\equiv j$, rA $\equiv K_i \equiv R_i$, rX $\equiv$ COUNT[i].

```
01  START ENT1 N          1    C1. Очистка COUNT.
02        STZ  COUNT,1    N    COUNT[i] ← 0.
```

Таблица 1

СОРТИРОВКА ПОСРЕДСТВОМ ПОДСЧЕТА (АЛГОРИТМ С)

КЛЮЧИ:	503	087	512	061	908	170	897	275	653	42 6	154	509	612	677	765	703
COUNT (нач.):	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
COUNT ($i = N$):	0	0	0	0	1	0	1	0	0	0	0	0	0	0	1	12
COUNT ($i = N-1$):	0	0	0	0	2	0	2	0	0	0	0	0	0	0	13	12
COUNT ($i = N-2$):	0	0	0	0	3	0	3	0	0	0	0	0	0	11	13	12
COUNT ($i = N-3$):	0	0	0	0	4	0	4	0	1	0	0	0	9	11	13	12
COUNT ($i = N-4$):	0	0	1	0	5	0	5	0	2	0	0	7	9	11	13	12
COUNT ($i = N-5$):	1	0	2	0	6	1	6	1	3	1	2	7	9	11	13	12
. . . . .	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
COUNT ($i = 2$):	6	1	8	0	15	3	14	4	10	5	2	7	9	11	13	12

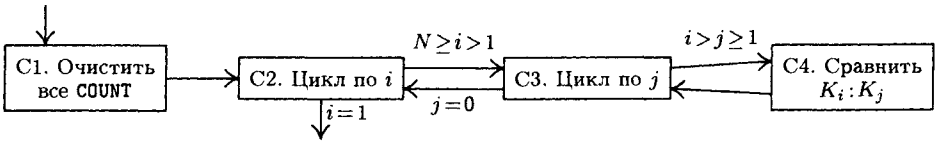


Рис. 8. Алгоритм С: подсчет сравнений.

```
03        DEC1 1          N
04        J1P  *-2        N     N ≥ i > 0.
05        ENT1 N          1     C2. Цикл по i.
06        JMP  1F         1
07  2H    LDA  INPUT,1    N-1
08        LDX  COUNT,1    N-1
09  3H    CMPA INPUT,2    A     C4. Сравнение K_i : K_j.
10        JGE  4F         A     Переход, если K_i ≥ K_j.
11        LD3  COUNT,2    B     COUNT[j]
12        INC3 1          B       +1
13        ST3  COUNT,2    B         → COUNT[j].
14        JMP  5F         B
15  4H    INCX 1          A-B   COUNT[i] ← COUNT[i] + 1.
16  5H    DEC2 1          A     C3. Цикл по j.
17        J2P  3B         A
18        STX  COUNT,1    N-1
19        DEC1 1          N-1
20  1H    ENT2 -1,1       N     N ≥ i > j > 0.
21        J2P  2B         N     ▌
```

Время работы этой программы равно $13N + 6A + 5B - 4$ машинных циклов, где N — число записей, A — число способов выбрать 2 предмета из N, т. е. $\binom{N}{2} = (N^2 - N)/2$, а B — число пар индексов, таких, что $j < i$ и $K_j > K_i$. Значит, B — число *инверсий* перестановки $K_1 \ldots K_N$; эта величина подробно анализировалась в разделе 5.1.1 (формулы 5.1.1–(12) и 5.1.1–(13)), в котором было найдено, что для неравных ключей, расположенных в случайном порядке,

$$B = \left(\min 0,\ \ \mathrm{ave}\ (N^2 - N)/4,\ \ \max\ (N^2 - N)/2,\ \ \mathrm{dev}\ \sqrt{N(N-1)(N+2.5)/6}\right).$$

Следовательно, выполнение программы C занимает от $3N^2 + 10N - 4$ до $5.5N^2 + 7.5N - 4$ машинных циклов, а среднее время работы находится посередине между этими крайними значениями. Например, для данных табл. 1 имеем $N = 16$, $A = 120$, $B = 41$, значит, программа C рассортирует их за время $1129u$. Модификация программы C, обладающей несколько иными временными характеристиками, рассматривается в упр. 5.

Множитель N^2, которым определяется время работы, свидетельствует о том, что алгоритм C неэффективен при большом N; при удвоении числа записей время увеличивается в четыре раза. Поскольку этот метод требует сравнения всех пар ключей (K_i, K_j), нет очевидного способа исключить зависимость от N^2. Тем не менее дальше в этой главе будет показано, что, используя другую методику, время сортировки для наихудшего случая можно уменьшить до порядка $N \log N$. Алгоритм C интересен для нас не эффективностью, а, главным образом, своей простотой; его описание служит примером того стиля, в котором будут описаны более сложные (и более эффективные) методы. Существует другая разновидность сортировки посредством подсчета, которая *действительно* весьма важна с точки зрения эффективности; она применима в основном в том случае, когда имеется много равных ключей и все они попадают в интервал $u \le K_j \le v$, ширина которого $(v - u)$ невелика. Кажется, что эти ограничения весьма сужают область применения такой модификации, но на самом деле мы увидим немало приложений этой идеи. Например, если применить этот метод к старшим цифрам ключей, а не ко всем ключам целиком, то файл окажется частично рассортированным, и будет уже сравнительно просто довести дело до конца.

Чтобы понять этот принцип, предположим, что все ключи лежат между 1 и 100. При первом просмотре файла можно подсчитать, сколько имеется ключей, равных 1, 2, ..., 100, а при втором — расположить записи в соответствующих местах области вывода. В следующем алгоритме (рис. 9) все это описано более подробно.

Алгоритм D (*Подсчет распределения*). Этот алгоритм сортирует записи $R_1, \ldots, R_N$, используя вспомогательную таблицу `COUNT[`u`]`, ..., `COUNT[`v`]` в предположении, что все ключи — целые числа в диапазоне $u \le K_j \le v$, $1 \le j \le N$. На последнем этапе выполнения алгоритма все записи в требуемом порядке переносятся в область вывода $S_1, \ldots, S_N$.

D1. [Сбросить счетчики.] Обнулить все счетчики `COUNT[`u`]`–`COUNT[`v`]`.

D2. [Цикл по j.] Выполнить шаг D3 при $1 \le j \le N$; затем перейти к шагу D4.

D3. [Увеличить `COUNT[`K_j`]`.] Увеличить значение `COUNT[`K_j`]` на 1.

D4. [Накопление.] (К этому моменту значение `COUNT[`i`]` есть число ключей, равных i.) Присвоить `COUNT[`i`]` $\leftarrow$ `COUNT[`i`]` + `COUNT[`$i - 1$`]` для $i = u + 1, u + 2, \ldots, v$.

D5. [Цикл по j.] (К этому моменту значение `COUNT[`i`]` есть число ключей, меньших или равных i, в частности `COUNT[`v`]` $= N$.) Выполнить шаг D6 при $j = N, N - 1, \ldots, 1$ и завершить выполнение процедуры.

D6. [Вывод R_j.] Присвоить $i \leftarrow$ `COUNT[`K_j`]`, $S_i \leftarrow R_j$ и `COUNT[`K_j`]` $\leftarrow i - 1$. ▮

Пример использования этого алгоритма рассмотрен в упр. 6; программу для `MIX` можно найти в упр. 9. При сформулированных выше условиях, т. е. когда диапазон $v - u$ мал, эта процедура сортировки работает очень быстро.

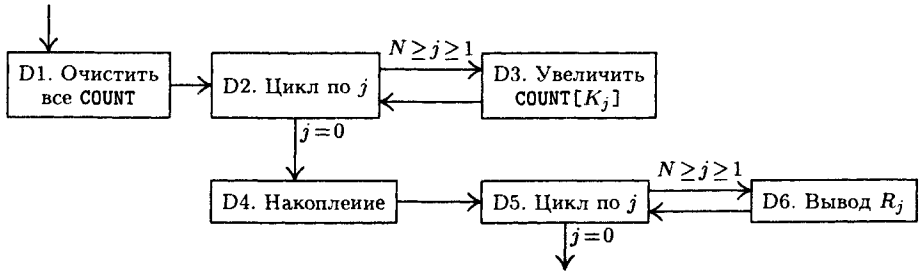


Рис. 9. Алгоритм D: подсчет распределения.

Сортировка посредством сравнения и подсчета, как в алгоритме C, впервые упоминается в работе E. H. Friend, *JACM* **3** (1956), 152, хотя ее автор и не утверждает, что это его собственное изобретение. Подсчет с распределением, как в алгоритме D, впервые разработан X. Сьювордом в 1954 году и использован при поразрядной сортировке, которую мы обсудим позже (см. раздел 5.2.5); этот метод также был опубликован под названием "Mathsort" в работе W. Feurzeig, *CACM* **3** (1960), 601.

УПРАЖНЕНИЯ

1. [*15*] Будет ли работать алгоритм C, если на шаге C2 значение i будет изменяться от 2 до N, а не от N до 2? Что произойдет, если на шаге C3 значение j будет изменяться от 1 до $i-1$?

2. [*21*] Покажите, что алгоритм C работает правильно и при наличии одинаковых ключей. Если $K_j = K_i$ и $j < i$, то где, в конце концов, окажется R_j: до или после R_i?

▶ **3.** [*21*] Будет ли алгоритм C работать правильно, если на шаге C4 заменить проверку "$K_i < K_j$" проверкой "$K_i \leq K_j$"?

4. [*16*] Напишите MIX-программу, которая "завершит" сортировку, начатую программой C; ваша программа должна поместить ключи в ячейки OUTPUT+1–OUTPUT+N в порядке возрастания. Сколько времени потребуется на это вашей программе?

5. [*22*] Станет ли программа C лучше в результате следующих изменений?

Ввести новую строку 08a: `INCX 0,2`
Изменить строку 10: `JGE 5F`
Изменить строку 14: `DECX 1`
Удалить строку 15.

6. [*18*] Промоделируйте вручную работу алгоритма D, показывая промежуточные результаты, получающиеся при сортировке 16 записей 5T, 0C, 5U, 0O, 9., 1N, 8S, 2R, 6A, 4A, 1G, 5L, 6T, 6I, 7O, 7N. Здесь цифры — это ключи, а буквы — сопутствующая информация в записях.

7. [*13*] Обеспечивает ли алгоритм D устойчивую сортировку?

8. [*15*] Будет ли алгоритм D работать правильно, если на шаге D5 значение j будет изменяться от 1 до N, а не от N до 1?

9. [*23*] Напишите MIX-программу для алгоритма D, аналогичную программе C и упр. 4. Выразите время работы программы в виде функции от N и $(v-u)$.

10. [*25*] Предложите эффективный алгоритм, который бы заменял N величин $(R_1, \ldots, R_N)$ соответственно величинами $(R_{p(1)}, \ldots, R_{p(N)})$, если даны значения $R_1, \ldots, R_N$ и перестановка $p(1) \ldots p(N)$ множества $\{1, \ldots, N\}$. Постарайтесь использовать как можно меньше

памяти. (Эта задача возникает, когда требуется перекомпоновать в памяти записи после сортировки таблицы адресов, не расходуя память на $2N$ записей.)

11. [*M27*] Напишите MIX-программу для алгоритма из упр. 10 и проанализируйте ее эффективность.

▶ **12.** [*25*] Предложите эффективный алгоритм перекомпоновки в памяти записей $R_1, \ldots, R_N$ в рассортированном порядке после завершения сортировки списка (см. рис. 7). Постарайтесь использовать минимум памяти.

▶ **13.** [*27*] Алгоритму D требуется память для $2N$ записей $R_1, \ldots, R_N$ и $S_1, \ldots, S_N$. Покажите, что можно обойтись памятью для N записей $R_1, \ldots, R_N$, если вместо шагов D5 и D6 использовать другую процедуру расстановки. (Таким образом, задача состоит в том, чтобы разработать алгоритм, который бы перекомпоновывал записи $R_1, \ldots, R_N$, основываясь на значениях COUNT[u], ..., COUNT[v] после выполнения шага D4, без использования дополнительной памяти; это, по существу, обобщение задачи, рассмотренной в упр. 10.)

5.2.1. Сортировка путем вставок

Упомянутый в начале раздела 5.2 способ, которым пользуются игроки в бридж, служит базовым для одного из важных семейств методов сортировки. Этот способ предполагает, что перед рассмотрением записи R_j предыдущие записи $R_1, \ldots, R_{j-1}$ уже упорядочены и R_j вставляется в соответствующее место. Приняв эту схему в качестве базовой, можно построить несколько любопытных ее модификаций.

Метод простых вставок. Простейший метод сортировки посредством вставок можно считать наиболее очевидным. Пусть $1 < j \le N$ и записи $R_1, \ldots, R_{j-1}$ уже размещены так, что

$$K_1 \le K_2 \le \cdots \le K_{j-1}.$$

(Напомним, что в этой главе через K_j обозначается ключ записи R_j.) Будем сравнивать по очереди K_j с K_{j-1}, K_{j-2}, ... до тех пор, пока не обнаружим, что запись R_j следует вставить между R_i и R_{i+1}; тогда подвинем записи $R_{i+1}, \ldots, R_{j-1}$ на одну позицию вверх и поместим новую запись в позицию $i+1$. Удобно совмещать операции сравнения и перемещения, перемежая их между собой, как продемонстрировано в следующем алгоритме; поскольку запись R_j как бы "погружается" на положенный ей уровень, этот способ часто называют *просеиванием* или *погружением* (рис. 10).

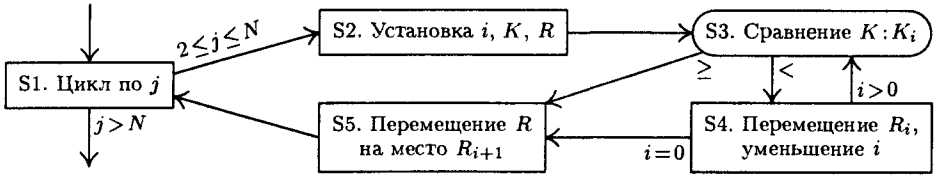


Рис. 10. Алгоритм S: сортировка методом простых вставок.

Алгоритм S (*Сортировка методом простых вставок*). Записи $R_1, \ldots, R_N$ переразмещаются на том же месте. После завершения сортировки их ключи будут упорядочены: $K_1 \le \cdots \le K_N$.

S1. [Цикл по j.] Выполнить шаги от S2 до S5 при $j = 2, 3, \ldots, N$ и после этого завершить выполнение процедуры.

S2. [Установка i, K, R.] Присвоить $i \leftarrow j-1$, $K \leftarrow K_j$, $R \leftarrow R_j$. (На последующих шагах будет предпринята попытка вставить запись R в нужное место, сравнивая K с K_i при убывающих значениях i.)

S3. [Сравнение $K : K_i$.] Если $K \geq K_i$, то перейти к шагу S5. (Мы нашли искомое место для записи R.)

S4. [Перемещение R_i, уменьшение i.] Присвоить $R_{i+1} \leftarrow R_i$, $i \leftarrow i-1$. Если $i > 0$, то вернуться к шагу S3. (Если $i = 0$, то K — наименьший из рассмотренных до сих пор ключей, а значит, запись R должна занять первую позицию.)

S5. [Перенос R на место R_{i+1}.] Присвоить $R_{i+1} \leftarrow R$. ▮

В табл. 1 показан процесс выполнения алгоритма S на множестве из шестнадцати чисел, которые используются для примеров в этом разделе. Данный метод чрезвычайно просто реализуется программно. На самом деле следующая MIX-программа самая короткая из всех "порядочных" программ сортировки в этой книге.

Таблица 1

ПРИМЕР СОРТИРОВКИ МЕТОДОМ ПРОСТЫХ ВСТАВОК

^503	: 087														
087	503	^: 512													
^087	503	512	: 061												
061	087	503	512	^: 908											
061	087	^503	512	908	: 170										
061	087	170	503	512	^908	: 897									
. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .
061	087	154	170	275	426	503	509	512	612	653	677	^765	897	908	: 703
061	087	154	170	275	426	503	509	512	612	653	677	703	765	897	908

Программа S (*Сортировка методом простых вставок*). Адреса записей, которые надо рассортировать, — от INPUT+1 до INPUT+N; записи сортируются в той же области памяти по ключу, занимающему целиком одно слово. Здесь rI1 $\equiv j - N$; rI2 $\equiv i$; rA $\equiv R \equiv K$; предполагается, что $N \geq 2$.

```
01  START ENT1 2-N          1            S1. Цикл по j. j ← 2.
02  2H    LDA  INPUT+N,1    N-1          S2. Установка i, K, R.
03        ENT2 N-1,1        N-1          i ← j-1.
04  3H    CMPA INPUT,2      B+N-1-A      S3. Сравнить K : K_i.
05        JGE  5F           B+N-1-A      Перейти к шагу S5, если K ≥ K_i.
06  4H    LDX  INPUT,2      B            S4. Переместить R_i, уменьшить i.
07        STX  INPUT+1,2    B            R_{i+1} ← R_i.
08        DEC2 1            B            i ← i-1.
09        J2P  3B           B            Перейти к шагу S3, если i > 0.
10  5H    STA  INPUT+1,2    N-1          S5. R поместить на место R_{i+1}.
11        INC1 1            N-1
12        J1NP 2B           N-1          2 ≤ j ≤ N. ▮
```

Время выполнения этой программы равно $9B+10N-3A-9$ машинных циклов*, где N — число сортируемых записей, A — число случаев, когда на шаге S4 значение i убывает до 0, а B — количество перемещений записей. Ясно, что A равно числу случаев, когда $K_j < \min(K_1, \ldots, K_{j-1})$ при $1 < j \le N$, т. е. это число левосторонних минимумов — величина, которая была подробно проанализирована в разделе 1.2.10. Нетрудно прийти к выводу, что B тоже известно: число перемещений записей при фиксированном j равно числу инверсий ключа K_j, так что B равно числу инверсий перестановки $K_1 K_2 \ldots K_N$. Следовательно, согласно формулам 1.2.10–(16), 5.1.1–(12) и 5.1.1–(13) имеем

$$A = \left(\min 0,\ \text{ave}\ H_N - 1,\ \max N - 1,\ \text{dev}\ \sqrt{H_N - H_N^{(2)}}\right),$$

$$B = \left(\min 0,\ \text{ave}\ (N^2 - N)/4,\ \max\ (N^2 - N)/2,\ \text{dev}\ \sqrt{N(N-1)(N+2.5)/6}\right),$$

а среднее время выполнения программы S в случае, если ключи различны и расположены в случайном порядке, равно $(2.25N^2 + 7.75N - 3H_N - 6)u$. В упр. 33 показано, как можно несколько повысить скорость.

Приведенные в качестве примера в табл. 1 данные содержат 16 элементов; в наборе имеется два левосторонних минимума, 087 и 061, и, как было показано в предыдущем разделе, 41 инверсия. Следовательно, $N = 16$, $A = 2$, $B = 41$, а общее время сортировки равно $514u$.

Бинарные и двухпутевые вставки. Когда при сортировке методом простых вставок обрабатывается j-я запись, ее ключ сравнивается в среднем примерно с $j/2$ ранее рассортированных ключей; поэтому общее число сравнений равно приблизительно $(1 + 2 + \cdots + N)/2 \approx N^2/4$, а это очень много, даже если N не так уж и велико. В разделе 6.2.1 будут проанализированы методы бинарного поиска, которые указывают, куда вставлять j-й элемент после приблизительно $\lg j$ операций сравнения с соответствующим образом выбранными элементами сортируемого набора. Например, если вставляется 64-я запись, можно сначала сравнить ключ K_{64} с K_{32}, а затем, если он меньше, сравнить его с K_{16}, если больше — с K_{48} и т. д., так что место для R_{64} будет найдено после всего лишь шести сравнений. Общее число сравнений для N вставляемых элементов равно приблизительно $N \lg N$, что существенно лучше, чем $\frac{1}{4}N^2$; в разделе 6.2.1 показано, что соответствующая программа не обязательно намного сложнее, чем программа для простых вставок. Этот метод называется методом *бинарных вставок*. Он упоминался Джоном Мочли (John Mauchly) еще в 1946 году, в первой публикации по машинной сортировке.

Неприятность состоит в том, что метод бинарных вставок позволяет решить задачу только наполовину: после того как найдено место, куда вставлять запись R_j, все равно нужно подвинуть примерно $\frac{1}{2}j$ ранее рассортированных записей, чтобы освободить место для R_j, так что общее время выполнения остается, по существу, пропорциональным N^2. В некоторых компьютерах (например, в IBM 705) используются процессоры, в наборе команд которых имеется и команда перемещения блока памяти, выполняемая аппаратно с большой скоростью. Но с ростом N зависимость от N^2, в конце концов, начинает преобладать. Например, анализ, проведенный

* В оригинале автор использует термин "unit" — единица, под которой подразумевается среднее время выполнения одной машинной команды (см. том 1). Это и есть не что иное, как длительность машинного цикла. — *Прим. перев.*

Х. Нэглером [H. Nagler, *CACM* **3** (1960), 618–620], показывает, что метод бинарных вставок не рекомендуется использовать при сортировке более $N = 128$ записей по 80 символов на компьютере IBM 705. Методика анализа применима и к другим компьютерам.

Разумеется, изобретательный программист может придумать какие-нибудь способы, позволяющие сократить число необходимых переписей в памяти; первый такой прием, предложенный в начале 50-х годов, проиллюстрирован в табл. 2. Здесь первый элемент помещается в середину области вывода и место для последующих элементов освобождается при помощи сдвигов влево или вправо, туда, куда удобнее. Таким образом удается сэкономить примерно половину времени работы по сравнению с использованием метода простых вставок за счет некоторого усложнения программы. Можно применять этот метод, используя не больше памяти, чем требуется для N записей (см. упр. 6), но мы не станем дольше задерживаться на таких "двухпутевых" вставках, так как разработаны и гораздо более интересные методы.

Таблица 2

ПРОЦЕСС СОРТИРОВКИ МЕТОДОМ ДВУХПУТЕВЫХ ВСТАВОК

			^	503			
		087		503	^		
		^	087	503	512		
	061	087		503	512	^	
	061	087	^	503	512	908	
061	087	170		503	512	^	908
061	087	170	^	503	512	897	908
061	087	170	275	503	512	897	908

Метод Шелла. Для алгоритма сортировки, который каждый раз перемещает запись только на одну позицию, среднее время выполнения будет в лучшем случае пропорционально N^2, потому что в процессе сортировки каждая запись должна пройти в среднем через $\frac{1}{3}N$ позиций (см. упр. 7). Поэтому, если желательно получить метод, существенно превосходящий по скорости метод простых вставок, необходим механизм, с помощью которого записи могли бы перемещаться большими скачками, а не короткими шажками.

Такой метод предложен в 1959 году Дональдом Л. Шеллом [Donald L. Shell, *CACM* **2**, 7 (July, 1959), 30–32] и известен во всем мире под именем своего автора. В табл. 3 проиллюстрирована общая идея, которая лежит в его основе. Сначала делим 16 записей на 8 групп по две записи в каждой группе: $(R_1, R_9), (R_2, R_{10}), \ldots, (R_8, R_{16})$. В результате сортировки каждой группы записей по отдельности приходим ко второй строке табл. 3. Этот процесс называется первым проходом. Обратите внимание на то, что элементы 154 и 512 поменялись местами, а 908 и 897 переместились вправо. Разделим теперь записи на четыре группы по четыре в каждой: $(R_1, R_5, R_9, R_{13}), \ldots, (R_4, R_8, R_{12}, R_{16})$. Затем опять рассортируем каждую группу в отдельности; результат этого второго прохода показан в третьей строке таблицы. На

Таблица 3

СОРТИРОВКА ШЕЛЛА СО СМЕЩЕНИЯМИ 8, 4, 2, 1

	503 087 512 061 908 170 897 275 653 426 154 509 612 677 765 703
Сортировка через 8:	
	503 087 154 061 612 170 765 275 653 426 512 509 908 677 897 703
Сортировка через 4:	
	503 087 154 061 612 170 512 275 653 426 765 509 908 677 897 703
Сортировка через 2:	
	154 061 503 087 512 170 612 275 653 426 765 509 897 677 908 703
Сортировка через 1:	
	061 087 154 170 275 426 503 509 512 612 653 677 703 765 897 908

третьем проходе сортируются две группы по восемь записей; процесс завершается четвертым проходом, во время которого сортируются все 16 записей. В каждой из промежуточных стадий сортировки участвуют либо сравнительно короткие массивы, либо уже сравнительно хорошо упорядоченные массивы, поэтому на каждом этапе можно пользоваться методом простых вставок и сортировка выполняется довольно быстро.

Метод сортировки Шелла также известен под именем Shellsort и метода сортировки с "убывающим смещением", поскольку каждый проход характеризуется смещением h, таким, что сортируются записи, каждая из которых отстоит от предыдущей на h позиций. Последовательность значений смещений 8, 4, 2, 1 не следует считать незыблемой; можно пользоваться *любой* последовательностью $h_{t-1}, h_{t-2}, \ldots, h_0$, но последнее смещение h_0 должно быть равно 1. Например, в табл. 4 продемонстрирована сортировка тех же данных со смещениями 7, 5, 3, 1. Как будет показано ниже, выбор значений смещений на последовательных проходах имеет весьма существенное значение для скорости сортировки.

Алгоритм D (*Сортировка Шелла*). Записи $R_1, \ldots, R_N$ перекомпоновываются в том же адресном пространстве памяти. После завершения сортировки их ключи будут упорядочены: $K_1 \le \cdots \le K_N$. Для управления процессом сортировки используется вспомогательная последовательность смещений $h_{t-1}, h_{t-2}, \ldots, h_0$, где $h_0 = 1$; правильно выбрав эти значения в последовательности, можно значительно сократить время сортировки. При $t = 1$ этот алгоритм сводится к алгоритму S.

D1. [Цикл по s.] Выполнить шаг D2 при $s = t-1, t-2, \ldots, 0$, после чего завершить процедуру.

D2. [Цикл по j.] Присвоить $h \leftarrow h_s$ и выполнить шаги от D3 до D6 при $h < j \le N$. (Для сортировки элементов, отстоящих один от другого на h позиций, воспользуемся методом простых вставок и в результате получим $K_i \le K_{i+h}$ для $1 \le i \le N-h$. Шаги от D3 до D6, по существу, такие же, как соответственно от S2 до S5 в алгоритме S.)

D3. [Установка i, K, R.] Присвоить $i \leftarrow j-h$, $K \leftarrow K_j$, $R \leftarrow R_j$.

D4. [Сравнение $K : K_i$.] Если $K \ge K_i$, то перейти к шагу D6.

Таблица 4

СОРТИРОВКА ШЕЛЛА СО СМЕЩЕНИЯМИ 7, 5, 3, 1

	503 087 512 061 908 170 897 275 653 426 154 509 612 677 765 703
Сортировка через 7:	
	275 087 426 061 509 170 677 503 653 512 154 908 612 897 765 703
Сортировка через 5:	
	154 087 426 061 509 170 677 503 653 512 275 908 612 897 765 703
Сортировка через 3:	
	061 087 170 154 275 426 512 503 653 612 509 765 677 897 908 703
Сортировка через 1:	
	061 087 154 170 275 426 503 509 512 612 653 677 703 765 897 908

D5. [Перемещение R_i, уменьшение i.] Присвоить $R_{i+h} \leftarrow R_i$, затем $i \leftarrow i - h$. Если $i > 0$, то возвратиться к шагу D4.

D6. [Перемещение R на место R_{i+h}.] Присвоить $R_{i+h} \leftarrow R$. ▮

Соответствующая программа для MIX не намного длиннее, чем наша программа для метода простых вставок. Строки 08–19 этой программы перенесены из программы S в более общий контекст алгоритма D.

Программа D (*Сортировка Шелла*). Предполагается, что смещения для сортировки хранятся во вспомогательной таблице и h_s находится по адресу $\mathtt{H} + s$; все смещения сортировки меньше N. Содержимое регистров таково: rI1 $\equiv j - N$; rI2 $\equiv i$; rA $\equiv R \equiv K$; rI3 $\equiv s$; rI4 $\equiv h$. Обратите внимание на то, что эта программа сама себя изменяет. Это сделано для того, чтобы добиться более эффективного выполнения внутреннего цикла.

```
01  START ENT3 T-1          1          D1. Цикл по s.  s ← t − 1.
02  1H    LD4  H,3          T          D2. Цикл по j.  h ← h_s.
03        ENT1 INPUT,4      T          Модифнкация адресов в трех
04        ST1  5F(0:2)      T             командах основного цикла.
05        ST1  6F(0:2)      T
06        ENN1 -N,4         T          rI1 ← N − h.
07        ST1  3F(0:2)      T
08        ENT1 1-N,4        T          j ← h + 1.
09  2H    LDA  INPUT+N,1    NT − S     D3. Присвоить i, K, R.
10  3H    ENT2 N-H,1        NT − S     i ← j − h.            [Измеияемая команда]
11  4H    CMPA INPUT,2      B+NT−S−A   D4. Сравнить K : K_i.
12        JGE  6F           B+NT−S−A   Перейти к шагу D6, если K ≥ K_i.
13        LDX  INPUT,2      B          D5. Переписать R_i, уменьшить i.
14  5H    STX  INPUT+H,2    B          R_{i+h} ← R_i.        [Изменяемая команда]
15        DEC2 0,4          B          i ← i − h.
16        J2P  4B           B          Перейти к шагу D4, если i > 0.
17  6H    STA  INPUT+H,2    NT − S     D6. Переместить R     [Изменяемая команда]
                                       на место R_{i+h}.
18  7H    INC1 1            NT − S     j ← j + 1.
19        J1NP 2B           NT − S     Перейти к шагу D3, если j ≤ N.
20        DEC3 1            T
21        J3NN 1B           T          t > s ≥ 0.  ▮
```

***Анализ метода Шелла.** Для рационального выбора последовательности значений смещений сортировки $h_{t-1}, \ldots, h_0$ для алгоритма D нужно проанализировать время выполнения как функцию от этих смещений. Такой анализ приводит к постановке очень красивых, но еще не до конца решенных математических задач; никому до сих пор не удалось найти наилучшую возможную последовательность смещений для больших N. Тем не менее известно довольно много интересных свойств сортировки методом Шелла с убывающим смещением, и мы здесь их кратко изложим; подробности будут рассмотрены в приведенных ниже упражнениях. [Читателям, не склонным к пространным математическим выкладкам, лучше пропустить следующие несколько страниц вплоть до начала обсуждения метода вставки в список, который следует за формулой (12).]

Счетчики частот выполнения в программе D показывают, что на время выполнения влияют пять факторов: размер массива N, число проходов (т. е. число различных смещений) $T = t$, сумма значений смещений в последовательности

$$S = h_0 + \cdots + h_{t-1},$$

число сравнений $B + NT - S - A$ и число перезаписей B. Как и при анализе программы S, здесь A равно числу левосторонних минимумов, встречающихся при промежуточных операциях сортировки, а B равно числу инверсий в подмассивах. Основным фактором, от которого зависит время выполнения, является величина B, поэтому на нее мы в основном и обратим свое внимание. При анализе будет предполагаться, что ключи различны и первоначально расположены в случайном порядке.

Назовем операцию шага D2 h-сортировкой. Тогда сортировка методом Шелла состоит из h_{t-1}-сортировки, за которой следует h_{t-2}-сортировка, ..., за которой следует h_0-сортировка. Массив, в котором $K_i \le K_{i+h}$ при $1 \le i \le N - h$, будем называть h-упорядоченным.

Рассмотрим сначала простейшее обобщение простых вставок, когда имеется всего два смещения: $h_1 = 2$ и $h_0 = 1$. Во время второго просмотра имеем 2-упорядоченную последовательность ключей $K_1 K_2 \ldots K_N$. Легко видеть, что число перестановок $a_1 a_2 \ldots a_n$ множества $\{1, 2, \ldots, n\}$, таких, что $a_i \le a_{i+2}$ при $1 \le i \le n - 2$, равно

$$\binom{n}{\lfloor n/2 \rfloor},$$

так как существует всего одна 2-упорядоченная перестановка для каждого выбора $\lfloor n/2 \rfloor$ элементов, расположенных в четных позициях $a_2 a_4 \ldots$; тогда остальные $\lceil n/2 \rceil$ элементов попадают в позиции с нечетными номерами. После 2-сортировки случайного массива с одинаковой вероятностью может получиться любая 2-упорядоченная перестановка. Каково среднее число инверсий во всех таких перестановках?

Пусть A_n — суммарное число инверсий во всех 2-упорядоченных перестановках множества $\{1, 2, \ldots, n\}$. Ясно, что $A_1 = 0$, $A_2 = 1$, $A_3 = 2$; рассмотрев шесть случаев

$$1324 \quad 1234 \quad 1243 \quad 2134 \quad 2143 \quad 3142,$$

находим, что $A_4 = 1 + 0 + 1 + 1 + 2 + 3 = 8$. Чтобы проанализировать A_n в общем случае, рассмотрим "решетчатую диаграмму" на рис. 11 для $n = 15$. На такой диаграмме 2-упорядоченную перестановку множества $\{1, 2, \ldots, n\}$ можно представить в виде пути из верхней левой угловой точки (0,0) в нижнюю правую угловую

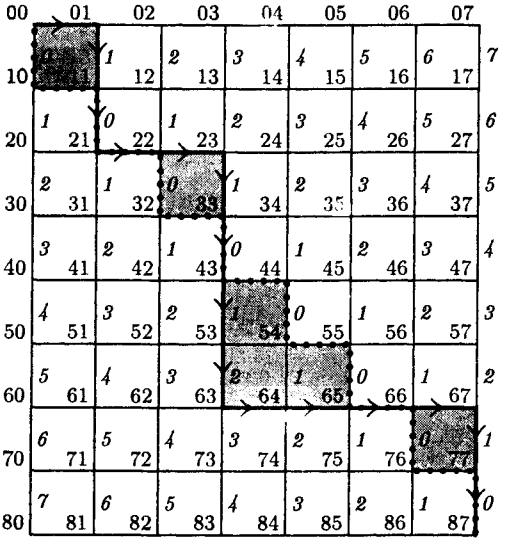


Рис. 11. Соответствие между 2-упорядочением и путями на решетке. Курсивом набраны веса, соответствующие числу инверсий в 2-упорядоченной перестановке.

точку $(\lceil n/2\rceil, \lfloor n/2\rfloor)$, если выполнять очередной k-й шаг пути вправо или вниз в соответствии с тем, где находится k: в четной или нечетной позиции перестановки. Этим правилом определяется взаимно однозначное соответствие между 2-упорядоченными перестановками и n-шаговыми путями из одного угла решетчатой диаграммы в другой. Например, изображенный на рис. 11 путь соответствует перестановке

$$2\ 1\ 3\ 4\ 6\ 5\ 7\ 10\ 8\ 11\ 9\ 12\ 14\ 13\ 15. \tag{1}$$

Далее, вертикальным отрезкам пути можно приписать веса, как показано на диаграмме; отрезку, ведущему из точки (i,j) в точку $(i+1,j)$, приписывается вес $|i-j|$. После несложных умозаключений читатель сможет убедиться в том, что сумма этих весов вдоль каждого пути равна числу инверсий в соответствующей перестановке; эта сумма также равна количеству заштрихованных квадратиков между данным путем и другим ступенчатым путем, выделенным на рисунке пунктирной линией с жирными точками (см. упр. 12). Так, например, перестановка (1) содержит $1+0+1+0+1+2+1+0=6$ инверсий.

Если $a \le a'$ и $b \le b'$, то число допустимых путей из (a,b) в (a',b') равно числу способов объединения $a'-a$ вертикальных отрезков с $b'-b$ горизонтальными, а именно

$$\binom{a'-a+b'-b}{a'-a}.$$

Следовательно, число перестановок, соответствующие пути которых проходят через вертикальный отрезок из (i,j) в $(i+1,j)$, равно

$$\binom{i+j}{i}\binom{n-i-j-1}{\lfloor n/2\rfloor-j}.$$

Умножая это значение на вес данного отрезка и суммируя по всем отрезкам, получаем

$$A_{2n} = \sum_{\substack{0\le i\le n\\ 0\le j\le n}} |i-j| \binom{i+j}{i}\binom{2n-i-j-1}{n-j};$$

$$A_{2n+1} = \sum_{\substack{0\le i\le n\\ 0\le j\le n}} |i-j| \binom{i+j}{i}\binom{2n-i-j}{n-j}. \tag{2}$$

Знаки абсолютной величины в этих суммах несколько усложняют вычисления, но в упр. 14 показано, что выражение для величины A_n имеет на удивление простой вид: $\lfloor n/2\rfloor 2^{n-2}$. Следовательно, среднее число инверсий в случайной 2-упорядоченной перестановке равно

$$\lfloor n/2\rfloor 2^{n-2} \Big/ \binom{n}{\lfloor n/2\rfloor}.$$

По формуле Стирлинга эта величина асимптотически приближается к $\sqrt{\pi/128}\,n^{3/2} \approx 0.15n^{3/2}$. Как легко видеть, максимальное число инверсий равно

$$\binom{\lfloor n/2\rfloor+1}{2} \approx \frac{1}{8}n^2.$$

Полезно более тщательно проанализировать распределение числа инверсий, рассмотрев производящие функции

$$\begin{aligned} h_1(z) &= 1,\\ h_2(z) &= 1+z,\\ h_3(z) &= 1+2z,\\ h_4(z) &= 1+3z+z^2+z^3, \qquad \ldots, \end{aligned} \tag{3}$$

как в упр. 15. Таким образом найдем, что стандартное отклонение тоже пропорционально $n^{3/2}$, так что число инверсий не слишком устойчиво распределено около среднего значения.

Рассмотрим теперь общий случай алгоритма D с двумя проходами, когда смещения сортировки равны h и 1.

Теорема H. *Среднее число инверсий в h-упорядоченной перестановке множества $\{1,2,\ldots,n\}$ равно*

$$f(n,h) = \frac{2^{2q-1}q!\,q!}{(2q+1)!}\left(\binom{h}{2}q(q+1) + \binom{r}{2}(q+1) - \frac{1}{2}\binom{h-r}{2}q\right), \tag{4}$$

где $q = \lfloor n/h\rfloor$ и $r = n \bmod h$.

Эта теорема принадлежит Дугласу Х. Ханту (Douglas H. Hunt), [Bachelor's thesis, Princeton University (April, 1967)]. Заметим, что формула справедлива и при $h \ge n$ и дает верный результат: $f(n,h) = \frac{1}{2}\binom{n}{2}$.

Доказательство. В h-упорядоченной перестановке содержится r упорядоченных подпоследовательностей длиной $q+1$ и $h-r$ подпоследовательностей длиной q. Каждая инверсия образуется из элементов двух различных подпоследовательно-

стей, а каждая пара различных упорядоченных подпоследовательностей в случайной h-упорядоченной перестановке определяет случайную 2-упорядоченную перестановку. Поэтому среднее число инверсий равно сумме средних значений числа инверсий во всех парах различных подпоследовательностей, а именно

$$\binom{r}{2}\frac{A_{2q+2}}{\binom{2q+2}{q+1}} + r(h-r)\frac{A_{2q+1}}{\binom{2q+1}{q}} + \binom{h-r}{2}\frac{A_{2q}}{\binom{2q}{q}} = f(n,h). \quad \blacksquare$$

Следствие. *Если последовательность смещений $h_{t-1}, \ldots, h_1, h_0$ удовлетворяет условию*

$$h_{s+1} \bmod h_s = 0 \qquad \text{при} \qquad t-1 > s \geq 0, \tag{5}$$

то среднее число операций перезаписи в алгоритме D равно

$$\sum_{t>s\geq 0} \bigl(r_s f(q_s+1, h_{s+1}/h_s) + (h_s - r_s) f(q_s, h_{s+1}/h_s)\bigr), \tag{6}$$

где $r_s = N \bmod h_s$, $q_s = \lfloor N/h_s \rfloor$, $h_t = Nh_{t-1}$, а функция f определяется формулой (4).

Доказательство. Процесс h_s-сортировки предусматривает сортировку методом простых вставок r_s (h_{s+1}/h_s)-упорядоченных подмассивов длиной q_s+1 и $(h_s - r_s)$ таких подмассивов длиной q_s. Поскольку предполагается, что исходная перестановка случайна и все ее элементы различны, то из условий делимости следует, что каждый из подмассивов — "случайная" (h_{s+1}/h_s)-упорядоченная перестановка в том смысле, что все (h_{s+1}/h_s)-упорядоченные перестановки равновероятны. $\blacksquare$

Условие (5) этого следствия всегда выполняется для *двухпроходной* сортировки методом Шелла, когда смещения равны соответственно h и 1. Пусть $q = \lfloor N/h \rfloor$, а $r = N \bmod h$, тогда среднее значение величины B в программе D равно

$$rf(q+1, N) + (h-r)f(q, N) + f(N,h) = \frac{r}{2}\binom{q+1}{2} + \frac{h-r}{2}\binom{q}{2} + f(N,h).$$

В первом приближении функция $f(n,h)$ равна $(\sqrt{\pi}/8)n^{3/2}h^{1/2}$; можно сравнить ее с гладкой кривой на рис. 12 при $n = 64$. Следовательно, время выполнения двухпроходной программы D примерно пропорционально

$$2N^2/h + \sqrt{\pi N^3 h}.$$

Поэтому наилучшее значение h равно приблизительно $\sqrt[3]{16N/\pi} \approx 1.72\sqrt[3]{N}$; при таком выборе h среднее время сортировки пропорционально $N^{5/3}$.

Таким образом, применяя метод Шелла и используя всего-навсего два прохода, можно существенно сократить время по сравнению с методом простых вставок с $O(N^2)$ до $O(N^{1.667})$. Ясно, что можно добиться лучших результатов, если использовать больше проходов. В упр. 18 обсуждается оптимальный выбор $h_{t-1}, \ldots, h_0$ при фиксированном t в случае, когда значения h ограничены условием делимости. Время выполнения при больших N сокращается до $O(N^{1.5+\epsilon/2})$, где $\epsilon = 1/(2^t - 1)$. Используя приведенные выше формулы, барьер $N^{1.5}$ преодолеть невозможно, потому что на последнем проходе в сумму инверсий всегда входит слагаемое

$$f(N, h_1) \approx (\sqrt{\pi}/8)N^{3/2}h_1^{1/2}.$$

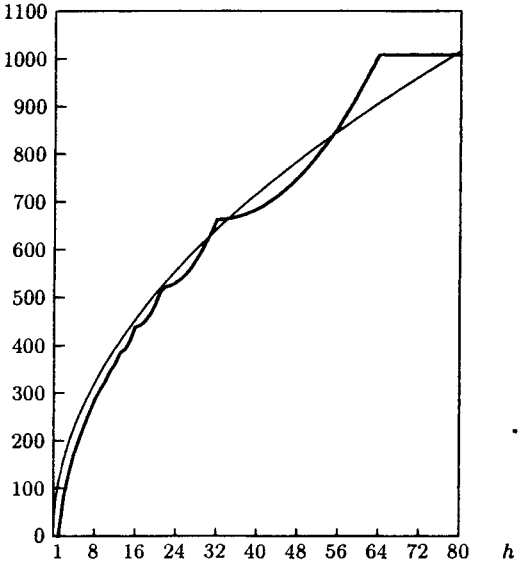


Рис. 12. Среднее число инверсий $f(n,h)$ в h-упорядоченном массиве из n элементов для случая $n = 64$.

Но интуиция подсказывает, что, если смещения $h_{t-1}, \ldots, h_0$ *не будут* удовлетворять условию делимости (5), можно достичь большего. Например, при последовательном выполнении 8-, 4- и 2-сортировок невозможно взаимодействие между ключами в четных и нечетных позициях; поэтому на долю заключительной 1-сортировки останется $\Theta(N^{3/2})$ инверсий. В то же время при последовательном выполнении 7-, 5- и 3-сортировок массив перекомпоновывается так, что при заключительной 1-сортировке не может встретиться более $2N$ инверсий (см. упр. 26)! На самом деле наблюдается удивительное явление.

Теорема K. *После h-сортировки k-упорядоченный массив остается k-упорядоченным.*

Таким образом, массив, который был сначала 7-рассортированным, а потом 5-рассортированным, становится одновременно 7- и 5-упорядоченным. А если подвергнуть его 3-сортировке, то полученный массив будет 7-, 5- и 3-упорядоченным. Примеры проявления этого замечательного свойства можно найти в табл. 4.

Доказательство. В упр. 20 показано, что эта теорема вытекает из следующей леммы.

Лемма L. *Пусть m, n, r — неотрицательные целые числа, а $(x_1, \ldots, x_{m+r})$ и $(y_1, \ldots, y_{n+r})$ — произвольные последовательности чисел, такие, что*

$$y_1 \le x_{m+1}, \qquad y_2 \le x_{m+2}, \qquad \ldots, \qquad y_r \le x_{m+r}. \tag{7}$$

Если последовательности x_k и y_k рассортировать независимо, так что $x_1 \le \cdots \le x_{m+r}$ и $y_1 \le \cdots \le y_{n+r}$, то соотношения (7) останутся в силе.

Доказательство. Известно, что все, кроме, быть может, m элементов последовательности x_k, превосходят (т. е. больше или равны) некоторые элементы последо-

вательности y, причем различные элементы x_k превосходят различные элементы y. Пусть $1 \le j \le r$. Так как после сортировки элемент x_{m+j} превосходит $m+j$ элементов x_k, то он превосходит, по крайней мере, j элементов y_k, а значит, он превосходит j *наименьших* элементов y_k. Следовательно, после сортировки имеем $x_{m+j} \ge y_j$. ▌ ▌

Из теоремы K видно, что при сортировке желательно пользоваться взаимно простыми значениями смещений, однако непосредственно из нее не следуют точные оценки числа перезаписей, выполняемых алгоритмом D. Так как число перестановок множества $\{1, 2, \ldots, n\}$, одновременно h- и k-упорядоченных, не всегда является делителем $n!$, то понятно, что теорема K объясняет далеко не все; в результате k- и h-сортировок некоторые k- и h-упорядоченные массивы получаются чаще других. Более того, анализ усредненного варианта алгоритма D для общего случая смещений $h_{t-1}, \ldots, h_0$ при $t > 3$ может поставить в тупик любого. Не существует даже очевидного способа отыскать "наихудший случай" для алгоритма D при произвольной последовательности смещений сортировки $(h_{t-1}, \ldots, h_0)$ и заданном N. Поэтому до сих пор все попытки анализа этого алгоритма в общем случае были тщетны. Но можно сделать определенные выводы из асимптотического поведения максимального времени сортировки, когда последовательность смещений имеет определенную форму.

Теорема P. *Если* $h_s = 2^{s+1} - 1$ *при* $0 \le s < t = \lfloor \lg N \rfloor$, то *время сортировки по алгоритму D* есть $O(N^{3/2})$.

Доказательство. Достаточно найти оценку B_s — числа перезаписей на s-м проходе, такую, чтобы $B_{t-1} + \cdots + B_0 = O(N^{3/2})$. Для первых $t/2$ просмотров при $t > s \ge t/2$ можно воспользоваться очевидной оценкой $B_s = O\bigl(h_s(N/h_s)^2\bigr)$, а для последующих проходов можно применить результат упр. 23: $B_s = O(Nh_{s+2}h_{s+1}/h_s)$. Следовательно, $B_{t-1} + \cdots + B_0 = O\bigl(N(2 + 2^2 + \cdots + 2^{t/2} + 2^{t/2} + \cdots + 2)\bigr) = O(N^{3/2})$. ▌

Эта теорема принадлежит А. А. Папернову и Г. В. Стасевичу [*Проблемы передачи информации*, **1**, 3 (1965), 81–98]. Она дает верхнюю оценку времени выполнения алгоритма в *худшем случае*, а не просто оценку *среднего* времени работы. Этот результат нетривиален, поскольку максимальное время выполнения в случае, когда смещения h удовлетворяют условию делимости (5), имеет порядок N^2, а в упр. 24 доказано, что показатель $3/2$ уменьшить нельзя.

Интересное улучшение по сравнению с теоремой P обнаружил в 1969 году Воган Пратт (Vaughan Pratt). *Если все смещения при сортировке выбираются из множества чисел вида* $2^p 3^q$, *меньших* N, *то время выполнения алгоритма D будет порядка* $N(\log N)^2$. В этом случае также можно внести в алгоритм несколько существенных упрощений. К сожалению, метод Пратта требует сравнительно большого числа проходов, так что это не лучший способ выбора смещений, если только N не очень велико (см. упр. 30 и 31). При реальных для практики значениях N наилучший набор значений смещений должен удовлетворять соотношению $h_s \approx \rho^s$, где параметр $\rho \approx h_{s+1}/h_s$ можно в первом приближении считать независимым от s, но он существенно зависит от N.

Из изложенного выше очевидно, что было бы неразумно выбирать значения смещений, которые являются множителем всех предшественников. Но в то же время нельзя делать заключение, что наилучшие значения смещений — взаимно простые числа всех предшественников. Конечно, каждый элемент массива, который уже

gh- и gk-рассортирован, причем $h \perp k$, имеет не более чем $\frac{1}{2}(h-1)(k-1)$ инверсий, когда наступает черед g-сортировки. (См. упр. 21.) Предложенная Праттом последовательность $\{2^p 3^q\}$ имеет при $N \to \infty$ преимущество именно вследствие этого свойства, но оно не очень сказывается на практике, поскольку нарастает это преимущество довольно медленно.

Дж. Инсерпи (J. Incerpi) и Р. Седгевик (R. Sedgewick) [см. *J. Comp. Syst. Sci.* **31** (1985), 210–224; а также *Lecture Notes in Comp. Sci.* **1136** (1996), 1–11] изобрели способ выбора лучшего варианта в том и другом смысле. Они показали, как сформировать последовательность смещений, для которых $h_s \approx \rho^s$, причем каждый очередной элемент в последовательности является наибольшим общим делителем своих предшественников. Задав любое значение $\rho > 1$, они далее определяли *базовую последовательность* $a_1, a_2, \ldots$, где a_k является наименьшим целым $\geq \rho^k$, таким, что $a_j \perp a_k$ при $1 \leq j < k$. Если, например, $\rho = 2.5$, базовая последовательность имеет вид

$$a_1, a_2, a_3, \ldots = 3, 7, 16, 41, 101, 247, 613, 1529, 3821, 9539, \ldots .$$

После этого определяются смещения, если положить $h_0 = 1$ и

$$h_s = h_{s-r} a_r \qquad \text{при } \binom{r}{2} < s \leq \binom{r+1}{2}. \tag{8}$$

Таким образом, начальный участок последовательности смещений принимает вид

$$1;\ a_1;\ a_2,\ a_1 a_2;\ a_1 a_3,\ a_2 a_3,\ a_1 a_2 a_3;\ \ldots .$$

Например, при $\rho = 2.5$ получим

$$1, 3, 7, 21, 48, 112, 336, 861, 1968, 4592, 13776, 33936, 86961, 198768, \ldots .$$

Решающим фактором в этом построении является возможность обратить рекуррентное соотношение (8):

$$h_s = h_{r+s}/a_r = h_{\binom{r}{2}}/a_{\binom{r}{2}-s} \qquad \text{при } \binom{r-1}{2} \leq s \leq \binom{r}{2}. \tag{9}$$

Таким образом, пользуясь аргументами из предыдущего абзаца, получим, что число инверсий на элемент для h_0-, h_1-сортировки и т. д. равно по крайней мере

$$b(a_2, a_1); b(a_3, a_2), b(a_3, a_1); b(a_4, a_3), b(a_4, a_2), b(a_4, a_1); \ldots, \tag{10}$$

где $b(h,k) = \frac{1}{2}(h-1)(k-1)$. Если $\rho^{t-1} \leq N < \rho^t$, общее число перезаписей B не превосходит суммы первых t этой последовательности, умноженной на N. Отсюда можно доказать, что в худшем случае время сортировки имеет порядок, значительно лучший, чем $N^{1.5}$ (см. упр. 41).

Теорема I. *Время выполнения алгоритма D равно $O(Ne^{c\sqrt{\ln N}})$, если смещения h_s определяются в соответствии с* (8). *Здесь $c = \sqrt{8 \ln \rho}$ и константы в O зависят от ρ.* ▌

Эта асимптотическая верхняя граница не имеет существенного значения при $N \to \infty$, поскольку последовательность, которую рекомендует Пратт, дает лучший результат. Рациональное же зерно теоремы I в том, что, если задано любое значение $\rho > 1$, последовательность смещений, имеющая практически показатель роста

Таблица 5

АНАЛИЗ АЛГОРИТМА D ПРИ $N = 8$

Смещения	$A_{\text{средн.}}$	$B_{\text{средн.}}$	S	T	Время на MIX
1	1.718	14.000	1	1	$204.85u$
2 1	2.667	9.657	3	2	$235.91u$
3 1	2.917	9.100	4	2	$220.15u$
4 1	3.083	10.000	5	2	$217.75u$
5 1	2.601	10.000	6	2	$209.20u$
6 1	2.135	10.667	7	2	$206.60u$
7 1	1.718	12.000	8	2	$209.85u$
4 2 1	3.500	8.324	7	3	$274.42u$
5 3 1	3.301	8.167	9	3	$253.60u$
3 2 1	3.320	7.829	6	3	$280.50u$

$h_s \approx \rho^s$, может иметь время сортировки, которое гарантированно равно $O(N^{1+\epsilon})$ при произвольных малых $\epsilon > 0$.

Рассмотрим, каким на практике может быть значение N, для чего проанализируем *общее* время выполнения программы D, а именно — $(9B + 10NT + 13T - 10S - 3A + 1)u$. В табл. 5 показано среднее время выполнения сортировки для различных последовательностей смещений при $N = 8$. Заметим, что при таком малом значении N в общем времени выполнения преобладают вспомогательные операции, поэтому наилучшие результаты получаются при $t = 1$; следовательно, при $N = 8$ лучше всего пользоваться методом простых вставок. (Среднее время выполнения программы S при $N = 8$ равно всего $191.85u$.) Любопытно, что наилучший результат в двухпроходном алгоритме достигается при $h_1 = 6$, поскольку большая величина S оказывается важнее малой величины B. Аналогично три смещения, 3 2 1, минимизируют среднее число перезаписей, но это не самый лучший набор значений для трехпроходной сортировки. Быть может, интересно привести некоторые "наихудшие" перестановки, максимизирующие число перезаписей, так как общий способ построения таких перестановок до сих пор не известен:

$h_2 = 5, \quad h_1 = 3, \quad h_0 = 1$: 8 5 2 6 3 7 4 1 (19 перезаписей)

$h_2 = 3, \quad h_1 = 2, \quad h_0 = 1$: 8 3 5 7 2 4 6 1 (17 перезаписей)

С ростом N наблюдается несколько иная картина. В табл. 6 показаны приближенные значения числа перемещений для различных последовательностей смещений при $N = 1000$. Первые несколько последовательностей удовлетворяют условию делимости (5), так что можно воспользоваться формулой (6) и упр. 19; для получения средних значений при других последовательностях смещений применялись эмпирические тесты. Были сгенерированы пять тысяч массивов по 1 000 случайных элементов, и каждый массив сортировался с каждой последовательностью смещений. Стандартное отклонение числа минимумов слева направо обычно составляло около 15, а стандартное отклонение числа перезаписей B обычно составляло около 300.

Эти данные позволяют выявить некоторые характеристики, но поведение алгоритма D все еще остается неясным. Шелл первоначально предполагал использовать смещения $\lfloor N/2 \rfloor$, $\lfloor N/4 \rfloor$, $\lfloor N/8 \rfloor$, ..., но это нежелательно, если двоичное представле-

Таблица 6

ДАННЫЕ О ВЫПОЛНЕНИИ АЛГОРИТМА D ПРИ $N = 1000$

Смещения											$A_{\text{средн.}}$	$B_{\text{средн.}}$	T
										1	6	249750	1
									17	1	65	41667	2
								60	6	1	158	26361	3
							140	20	4	1	262	21913	4
						256	64	16	4	1	362	20459	5
					576	192	48	16	4	1	419	20088	6
				729	243	81	27	9	3	1	378	18533	7
		512	256	128	64	32	16	8	4	2	1	493	16435	10
			500	250	125	62	31	15	7	3	1	516	7655	9
			501	251	125	63	31	15	7	3	1	558	7370	9
			511	255	127	63	31	15	7	3	1	559	7200	9
				255	127	63	31	15	7	3	1	436	7445	8
					127	63	31	15	7	3	1	299	8170	7
						63	31	15	7	3	1	190	9860	6
							31	15	7	3	1	114	13615	5
		513	257	129	65	33	17	9	5	3	1	561	6745	10
			257	129	65	33	17	9	5	3	1	440	6995	9
				129	65	33	17	9	5	3	1	304	7700	8
					65	33	17	9	5	3	1	197	9300	7
						33	17	9	5	3	1	122	12695	6
		683	341	171	85	43	21	11	5	3	1	511	7365	10
			341	171	85	43	21	11	5	3	1	490	7490	9
						255	63	15	7	3	1	373	8620	6
						257	65	17	5	3	1	375	8990	6
						341	85	21	5	3	1	410	9345	6
377	233	144	89	55	34	21	13	8	5	3	2	1	518	7400	13
	233	144	89	55	34	21	13	8	5	3	2	1	432	7610	12
						377	144	55	21	8	3	1	456	8795	7
						365	122	41	14	5	2	1	440	8085	7
							364	121	40	13	4	1	437	8900	6
								121	40	13	4	1	268	9790	5
						336	112	48	21	7	3	1	432	7840	7
			306	170	90	45	18	10	5	2	1	465	6755	9
						169	91	49	13	7	1	349	8698	6
				275	125	121	55	25	11	5	1	446	6788	8
				929	505	209	109	41	19	5	1	512	7725	8
					505	209	109	41	19	5	1	519	7790	7
						209	109	41	19	5	1	382	8165	6

ние числа N содержит длинные цепочки нулей. Лазарус и Фрэнк [Lazarus и Frank, *CACM* **3** (1960), 20–22] предложили использовать, по существу, ту же последовательность, но добавляя 1 там, где это необходимо, чтобы сделать все смещения нечетными. Хиббард [Hibbard, *CACM* **6** (1963), 206–213] предложил использовать смещения вида $2^k - 1$; Папернов и Стасевич предложили последовательность $2^k + 1$. Среди других естественных последовательностей, использованных для получения

табл. 6, — последовательности $(2^k - (-1)^k)/3$ и $(3^k - 1)/2$, числа Фибоначчи и предложенная Инсерпи и Седгевиком последовательность (8) для $\rho = 2.5$ и $\rho = 2$. Также включены последовательности $\{5^p 11^q\}$ и $\{7^p 13^q\}$, аналогичные предложенной Праттом, поскольку они сохраняют асимптотический характер сходимости к $O(N(\log N)^2)$, но приводят к более низким накладным расходам при малых N. Последний вариант в табл. 6 исходит из другой последовательности, предложенной все тем же Седгевиком, но основанной на несколько измененной эвристике [см. *J. Algorithms* **7** (1986), 159–173]:

$$h_s = \begin{cases} 9 \cdot 2^s - 9 \cdot 2^{s/2} + 1, & \text{если } s \text{ четно;} \\ 8 \cdot 2^s - 6 \cdot 2^{(s+1)/2} + 1, & \text{если } s \text{ нечетно.} \end{cases} \tag{11}$$

Седгевик доказал, что, когда используются сформированные по этому правилу смещения $(h_0, h_1, h_2, \ldots) = (1, 5, 19, 41, 109, 209, \ldots)$, время выполнения в худшем случае не превышает $O(N^{4/3})$.

Минимальное число перемещений, 6 600, наблюдается для смещений вида $2^k + 1$, а также в последовательностях Инсерпи и Седгевика при $\rho = 2$. Но важно понимать, что надо учитывать не только число перезаписей, хотя именно оно асимптотически доминирует в общем времени работы. Так как время выполнения программы D равно $9B + 10NT + \cdots$ единиц (машинных циклов), ясно, что экономия одного прохода примерно эквивалентна сокращению числа перезаписей на $\frac{10}{9}N$; при $N = 1000$ целесообразнее добавить 1 111 перезаписей, если за счет этого удастся сэкономить один проход. Поэтому представляется неразумным начинать с h_{t-1}, большего, чем, скажем, $\frac{1}{3}N$, поскольку большое значение смещения не убавит числа последующих перезаписей настолько, чтобы оправдать первый проход.

Обширное экспериментальное тестирование, выполненное М. А. Вайсом (M. A. Weiss) [*Comp. J.* **34** (1991), 88–91], четко выявило, что среднее число перезаписей, выполняемых в ходе сортировки по алгоритму D при смещениях $2^k - 1, \ldots, 15, 7, 3, 1$, примерно пропорционально $N^{5/4}$. Если быть более точным, Вайс показал, что $B_{\text{ave}} \approx 1.55N^{5/4} - 4.48N + O(N^{3/4})$ при $100 \le N \le 12000000$, если используются эти смещения; эмпирическое стандартное отклонение составляло примерно $.065N^{5/4}$. Он также выяснил, что последовательность Седгевика (11) приводит к асимптотически более высокой производительности, $B_{\text{ave}} \approx 0.43N^{7/6} + 18.5N + O(N^{5/6})$. Удивительно, но стандартное отклонение для этой последовательности смещений оказалось довольно мало, примерно $N^{3/4}$.

В табл. 7 показаны типовые характеристики числа перезаписей, отнесенные к одному проходу, которые были получены в трех экспериментах со случайными данными. Использовалась последовательность смещений вида $2^k - 1$, $2^k + 1$ и (11). В каждом эксперименте использовались те же массивы данных. Общее число перезаписей $\sum_s B_s$ в этих экспериментах оказалась равным соответственно 346 152, 329 532 и 248 788, так что первенство в этих "соревнованиях" следует отдать последовательности (11).

Хотя с каждым новым экспериментом мы все глубже проникаем в тайны сортировки по алгоритму D, три десятилетия исследований не дали окончательного ответа на главный вопрос — какая последовательность смещений наилучшая. Если N меньше 1 000, самое простое правило наподобие

$$\textit{Положить } h_0 = 1,\ h_{s+1} = 3h_s + 1 \textit{ и остановиться на } h_{t-1} \textit{ при } 3h_t \ge N \tag{12}$$

Таблица 7

КОЛИЧЕСТВО ПЕРЕЗАПИСЕЙ НА ПРОХОД: ЭКСПЕРИМЕНТЫ ПРИ $N = 20000$

h_s	B_s	h_s	B_s	h_s	B_s
4095	19458	4097	19459	3905	20714
2047	15201	2049	14852	2161	13428
1023	16363	1025	15966	929	18206
511	18867	513	18434	505	16444
255	23232	257	22746	209	21405
127	28034	129	27595	109	19605
63	33606	65	34528	41	26604
31	40350	33	45497	19	23441
15	66037	17	48717	5	38941
7	43915	9	38560	1	50000
3	24191	5	20271		
1	16898	3	9448		
		1	13459		

оказывается не хуже любого другого. Но для больших значений N можно рекомендовать последовательность Седгевика (11), которая обрывается на h_{t-1}, если $3h_t \geq N$.

В упр. 43 продемонстрировано, как удалить проверку $i > 0$ на шаге D5. Это изменение позволяет ускорить сортировку примерно на 10%.

Вставки в список. Оставим теперь метод Шелла и рассмотрим другие пути усовершенствования метода простых вставок. Среди общих способов улучшения алгоритма один из самых важных основывается на тщательном анализе структур данных, поскольку реорганизация структур данных, позволяющая избежать ненужных операций, часто дает существенный эффект. Дальнейшее обсуждение этой общей идеи можно найти в разделе 2.4, в котором рассматривается довольно сложный алгоритм. Посмотрим, как она применяется к такому нехитрому алгоритму, как простые вставки. Какова наиболее подходящая структура данных для алгоритма S?

Сортировка методом простых вставок состоит из двух основных операций:

i) просмотра упорядоченного массива для нахождения наибольшего ключа, меньшего или равного данному ключу;

ii) вставки новой записи в определенное место упорядоченного массива.

Массив — это, очевидно, линейный список, и алгоритм S обрабатывает его, используя последовательное распределение (раздел 2.2.2); поэтому для выполнения каждой операции вставки необходимо переместить примерно половину записей. С другой стороны, нам известно, что для вставок идеально подходит хранение данных в памяти в виде связного списка (раздел 2.2.3), так как при этом требуется изменить лишь несколько связей; другая операция — последовательный просмотр — при использовании связного списка почти так же проста, как и при последовательном размещении данных. Поскольку списки всегда просматриваются в одном и том же направлении, достаточно иметь списки с однонаправленной связью. Таким образом, приходим к выводу, что "правильная" структура данных для метода простых вставок — линейные списки с однонаправленными связями. Целесообразно также изменить алгоритм S, чтобы список просматривался в порядке возрастания.

Алгоритм L (*Метод вставки в список*). Предполагается, что записи $R_1, \ldots, R_N$ содержат ключи $K_1, \ldots, K_N$ и поля связи $L_1, \ldots, L_N$, в которых могут храниться числа от 0 до N; имеется также еще одно поле связи L_0 в некоторой искусственной записи R_0 в начале массива. Алгоритм устанавливает поля связи так, что записи оказываются связанными в порядке возрастания. Так, если $p(1) \ldots p(N)$ — "устойчивая" перестановка, такая, что $K_{p(1)} \le \cdots \le K_{p(N)}$, то в результате применения алгоритма получим

$$L_0 = p(1); \qquad L_{p(i)} = p(i+1) \quad \text{при} \quad 1 \le i < N; \qquad L_{p(N)} = 0. \tag{13}$$

L1. [Цикл по j.] Присвоить $L_0 \leftarrow N$, $L_N \leftarrow 0$. (L_0 служит "головным" элементом списка, а 0 — пустой связью; следовательно, список, по существу, циклический.) Выполнить шаги от L2 до L5 при $j = N-1, N-2, \ldots, 1$ и завершить процедуру.

L2. [Установка p, q, K.] Присвоить $p \leftarrow L_0$, $q \leftarrow 0$, $K \leftarrow K_j$. (На последующих шагах запись R_j будет вставлена в нужное место в связном списке путем сравнения ключа K с предыдущими ключами в порядке возрастания. Переменные p и q служат указателями на текущее место в списке, причем $p = L_q$, так что q всегда на один шаг отстает от p.)

L3. [Сравнение $K : K_p$.] Если $K \le K_p$, то перейти к шагу L5. (Найдено искомое положение записи R в списке между R_q и R_p.)

L4. [Продвинуть p, q.] Присвоить $q \leftarrow p$, $p \leftarrow L_q$. Если $p > 0$, то возвратиться к шагу L3. (Если $p = 0$, то K — наибольший ключ, обнаруженный до сих пор; следовательно, запись R должна попасть в конец списка, между R_q и R_0.)

L5. [Вставка в список.] Присвоить $L_q \leftarrow j$, $L_j \leftarrow p$. ▮

Этот алгоритм важен не только потому, что он является простым методом сортировки, но и потому, что он часто входит в состав других алгоритмов обработки списков. В табл. 8 показано несколько первых шагов сортировки шестнадцати чисел, выбранных нами для примеров.

Таблица 8

ПРИМЕР РЕАЛИЗАЦИИ МЕТОДА ВСТАВКИ В СПИСОК

j:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
K_j:	–	503	087	512	061	908	170	897	275	653	426	154	509	612	677	765	703
L_j:	16	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	0
L_j:	16	–	–	–	–	–	–	–	–	–	–	–	–	–	–	0	15
L_j:	14	–	–	–	–	–	–	–	–	–	–	–	–	–	16	0	15
. . .	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.

Программа L (*Метод вставки в список*). Предполагается, что ключ K_j хранится по адресу `INPUT`$+j$(0:3), а L_j хранится по адресу `INPUT`$+j$(4:5). Значения в регистрах таковы: rI1 $\equiv j$; rI2 $\equiv p$; rI3 $\equiv q$; rA(0:3) $\equiv K$.

```
01 KEY   EQU  0:3
02 LINK  EQU  4:5
```

03	START	ENT1	N	1	*L1. Цикл по j.* $j \leftarrow N$.
04		ST1	INPUT(LINK)	1	$L_0 \leftarrow N$.
05		STZ	INPUT+N(LINK)	1	$L_N \leftarrow 0$.
06		JMP	6F	1	Переход на уменьшение j.
07	2H	LD2	INPUT(LINK)	$N-1$	*L2. Установка p, q, K.* $p \leftarrow L_0$.
08		ENT3	0	$N-1$	$q \leftarrow 0$.
09		LDA	INPUT,1	$N-1$	$K \leftarrow K_j$.
10	3H	CMPA	INPUT,2(KEY)	$B+N-1-A$	*L3. Сравнить $K : K_p$.*
11		JLE	5F	$B+N-1-A$	Перейти к шагу L5, если $K \le K_p$.
12	4H	ENT3	0,2	B	*L4. Продвинуть p, q.* $q \leftarrow p$.
13		LD2	INPUT,3(LINK)	B	$p \leftarrow L_q$.
14		J2P	3B	B	Перейти к шагу L3, если $p > 0$.
15	5H	ST1	INPUT,3(LINK)	$N-1$	*L5. Вставить в список.* $L_q \leftarrow j$.
16		ST2	INPUT,1(LINK)	$N-1$	$L_j \leftarrow p$.
17	6H	DEC1	1	N	
18		J1P	2B	N	$N > j \ge 1$. ▌

Время выполнения этой программы равно $7B + 14N - 3A - 6$ машинных циклов, где N — длина массива, $A + 1$ — число правосторонних максимумов, а B — число инверсий в исходной перестановке. (Ср. с анализом программы S. Обратите внимание, что программа L не перемещает записи в памяти; это можно сделать, как в упр. 5.2–12, затратив дополнительно $20N$ машинных циклов.) Программа S требует $(9B + 10N - 3A - 9)u$, а так как B равно примерно $\frac{1}{4}N^2$, то нетрудно видеть, что за счет дополнительного пространства памяти, выделенного для полей связи, удалось сэкономить примерно 22% времени выполнения. Тщательно продумав программу, можно сберечь еще 22% (см. упр. 33), но время выполнения все же останется пропорциональным N^2.

Подведем итог сделанному. Мы начали с алгоритма S — простого и очевидного алгоритма сортировки, который выполняет около $\frac{1}{4}N^2$ сравнений и около $\frac{1}{4}N^2$ перемещений записей данных в памяти. Этот алгоритм был усовершенствован путем использования бинарных вставок, при которых выполняется около $N \lg N$ сравнений и $\frac{1}{4}N^2$ перемещений записей данных в памяти. Несколько изменив структуру данных, применив двухпутевые вставки, нам удалось сократить число перезаписей до $\frac{1}{8}N^2$. При сортировке методом Шелла с убывающим смещением число сравнений и перезаписей снижается примерно до $N^{7/6}$ для тех значений N, которые встречаются на практике; при $N \to \infty$ это число можно сократить до порядка $N(\log N)^2$. Дальнейшее стремление улучшить алгоритм S с помощью связной структуры данных привело нас к методу вставки в список, который требует около $\frac{1}{4}N^2$ сравнений, 0 перезаписей и $2N$ операций изменения связи.

Можно ли объединить лучшие свойства этих методов, т. е. сократить число сравнений до порядка $N \log N$, как при бинарных вставках, и в то же время исключить перемещения записей в памяти, как при вставках в список? Ответ утвердительный — нужно перейти к древовидной структуре данных. Такой вариант впервые был исследован в 1957 году Д. Дж. Уилером (D. J. Wheeler), который предложил использовать двухпутевые вставки до тех пор, пока не возникнет необходимость перемещать записи в памяти. Вместо операции перезаписи он предложил вставлять указатель на новую область памяти и рекурсивно применять ту же процедуру ко всем элементам, которые должны вставляться в эту новую область памяти. Ориги-

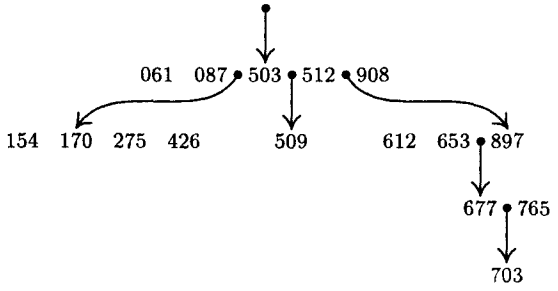


Рис. 13. Пример схемы Уилера для вставок в дерево.

нальный метод Уилера [см. A. S. Douglas, *Comp. J.* **2** (1959), 5] предполагал использование довольно замысловатой комбинации последовательной и связной памяти с узлами переменного размера; для тех шестнадцати чисел, которые используются в качестве примера неупорядоченной последовательности, таким способом было бы сформировано дерево, показанное на рис. 13. Аналогичную, но более простую схему со вставками в бинарную древовидную структуру изобрел К. М. Бернерс-Ли (C. M. Berners-Lee) примерно в 1958 году [см. *Comp. J.* **3** (1960), 174, 184]. Поскольку сам метод с использованием бинарного дерева и его модификации очень важны как для сортировки, так и для поиска, его подробное обсуждение мы отложим до раздела 6.2.2 главы 6.

Еще один путь улучшения метода простых вставок — попытаться вставлять несколько записей одновременно. Если, например, имеется массив из 1 000 элементов и 998 из них уже рассортированы, то алгоритм S выполнит еще два просмотра массива (вставив сначала R_{999}, а потом R_{1000}). Очевидно, можно сэкономить время, если сначала сравнить ключи K_{999} и K_{1000} и выяснить, какой из них больше, а затем вставить *оба* ключа за один просмотр массива. Комбинированная операция такого рода требует около $\frac{2}{3}N$ сравнений и перезаписей (см. упр. 3.4.2–5) вместо двух просмотров, примерно по $\frac{1}{2}N$ сравнений и перезаписей каждый.

Иначе говоря, обычно полезно "группировать" операции, которые требуют длительного поиска, чтобы можно было выполнить несколько операций сразу. Если довести эту идею до ее естественного завершения, то будет заново открыта сортировка посредством слияния, настолько важная, что ей посвящен отдельный раздел (5.2.4).

Сортировка с вычислением адреса. Ну уж теперь-то, несомненно, исчерпаны все возможные способы усовершенствования методов простых вставок! Но давайте подумаем еще. Представьте себе, что вам поручили расставить несколько дюжин книг на полке по фамилиям авторов, причем книги берутся с полки в случайном порядке. Ставя книгу на полку, вы, естественно, попробуете прикинуть, где примерно она будет, в конце концов, стоять, и таким образом потенциально сократите число необходимых в будущем сравнений и перестановок. Эффективность процесса повысится, если на полке будет немного больше места, чем это абсолютно необходимо. Такой метод для реализации компьютерной сортировки впервые предложили Исаак (Isaac) и Синглтон (Singleton) [см. *JACM* **3** (1956), 169–174]; он получил дальнейшее развитие в работе Тартера (Tarter) и Кронмэла (Kronmal) [см. *Proc. ACM Nat'l Conf.* **21** (1966), 331–337].

Сортировка с вычислением адреса обычно требует дополнительного пространства в памяти либо для того, чтобы оставить достаточно свободного места и не делать много лишних перемещений, либо для хранения вспомогательных таблиц, которые позволяли бы учитывать неравномерность распределения ключей. (См. сортировку методом подсчета распределения (алгоритм 5.2D), которая является разновидностью сортировки с вычислением адреса.) Дополнительное пространство будет, по-видимому, использоваться наилучшим образом, если отвести его для полей связи, как в методе вставок в список. К тому же отпадает необходимость в выделении отдельных областей для ввода и вывода; все операции можно выполнить в одной и той же области памяти. Основная цель — так обобщить метод вставок в список, чтобы работать не с одним списком, а с несколькими. Каждый список содержит ключи из определенного диапазона. Мы делаем важное предположение о том, что ключи распределены довольно равномерно и не скапливаются хаотически в отдельных областях значений. Множество всех возможных значений ключей разбивается на M частей, и предполагается, что данный ключ попадает в данное подмножество с вероятностью $1/M$. Отводим дополнительную память для головных элементов M списков, а каждый список обрабатываем, как при простых вставках в список.

Нет необходимости приводить здесь этот алгоритм со всеми подробностями. Достаточно вначале установить все головные элементы списков равными Λ, для каждого вновь поступившего элемента предварительно решить, в какое из M подмножеств он попадает, после чего вставить его в соответствующий список, как в алгоритме L.

Чтобы проиллюстрировать этот метод в действии, предположим, что наши 16 ключей разделены на $M = 4$ поддиапазона: 0–249, 250–499, 500–749, 750–999. В процессе сортировки по мере того, как будут вставляться ключи $K_1, K_2, \ldots, K_{16}$, получатся следующие конфигурации.

	Пришли 4 элемента	Пришли 8 элементов	Пришли 12 элементов	Всего
Список 1:	061, 087	061, 087, 170	061, 087, 154, 170	061, 087, 154, 170
Список 2:		275	275, 426	275, 426
Список 3:	503, 512	503, 512	503, 509, 512, 653	503, 509, 512, 612, 653, 677, 703
Список 4:		897, 908	897, 908	765, 897, 908

(Программа M, описанная ниже, в действительности вставляет ключи в обратном порядке, $K_{16}, \ldots, K_2, K_1$, но окончательный результат тот же.) Благодаря организации в памяти структуры связных списков не возникает проблем, связанных с выделением памяти при изменении длины списка. При желании в конце выполнения программы все списки можно объединить в один (см. упр. 35).

Программа M (*Вставка в несколько списков*). При разработке этой программы сделано такое же предположение, как и в программе L, но ключи должны быть *неотрицательными*, т. е.

$$0 \le K_j < (\texttt{BYTESIZE})^3.$$

В программе этот диапазон делится на `M` равных частей посредством умножения каждого ключа на соответствующую константу. Головные элементы списков хранятся в ячейках от `HEAD+1` до `HEAD+M`.

```
01 KEY   EQU  1:3
02 LINK  EQU  4:5
03 START ENT2 M                  1
04       STZ  HEAD,2             M      HEAD[p] ← Λ.
05       DEC2 1                  M
06       J2P  *-2                M      M ≥ p ≥ 1.
07       ENT1 N                  1      j ← N.
08 2H    LDA  INPUT,1(KEY)       N
09       MUL  =M(1:3)=           N      rA ← ⌊M · K_j/BYTESIZE^3⌋.
10       STA  *+1(1:2)           N
11       ENT4 0                  N      rI4 ← rA.
12       ENT3 HEAD+1-INPUT,4     N      q ← LOC(HEAD[rA]).
13       LDA  INPUT,1            N      K ← K_j.
14       JMP  4F                 N      Переход на установку p.
15 3H    CMPA INPUT,2(KEY)     B+N-A
16       JLE  5F               B+N-A    Переход на вставку, если K ≤ K_p.
17       ENT3 0,2                B      q ← p.
18 4H    LD2  INPUT,3(LINK)    B+N      p ← LINK(q).
19       J2P  3B               B+N      Переход, если не конец списка.
20 5H    ST1  INPUT,3(LINK)      N      LINK(q) ← LOC(R_j).
21       ST2  INPUT,1(LINK)      N      LINK(LOC(R_j)) ← p.
22 6H    DEC1 1                  N
23       J1P  2B                 N      N ≥ j ≥ 1.  ▌
```

Эта программа написана для любого значения M, но лучше зафиксировать M, положив его равным некоторому приемлемому значению; можно, например, положить $M = \texttt{BYTESIZE}$, тогда головные элементы списков можно очистить с помощью одной-единственной команды `MOVE`, а последовательность команд 08–11, реализующих умножение, заменить единственной командой `LD3 INPUT,1(1:1)`. Наиболее заметное отличие программы M от программы L состоит в том, что в программе M нужно принимать во внимание и возможность образования пустого списка, когда не надо делать сравнений.

Сколько же времени мы экономим, имея M списков вместо одного? Общее время выполнения программы M равно $7B+31N-3A+4M+2$ машинных циклов, где M — число списков, N — число сортируемых записей, A и B равны соответственно числу правосторонних максимумов и числу инверсий среди ключей, принадлежащих каждому списку. (Анализируя этот алгоритм, в отличие от всех других в данном разделе, мы включаем в подсчет A и крайний справа элемент непустой перестановки, а не игнорируем его.) Мы уже проанализировали величины A и B при $M = 1$, когда их средние значения оказались равными соответственно H_N и $\frac{1}{2}\binom{N}{2}$. Согласно предположению о распределении ключей вероятность того, что данный список в конце сортировки будет содержать ровно n элементов, есть "биномиальная" вероятность

$$\binom{N}{n}\left(\frac{1}{M}\right)^n\left(1-\frac{1}{M}\right)^{N-n}. \tag{14}$$

Поэтому средние значения величин A и B в общем случае равны

$$A_{\text{ave}} = M\sum_n \binom{N}{n}\left(\frac{1}{M}\right)^n\left(1-\frac{1}{M}\right)^{N-n} H_n; \tag{15}$$

$$B_{\text{ave}} = M \sum_n \binom{N}{n} \left(\frac{1}{M}\right)^n \left(1 - \frac{1}{M}\right)^{N-n} \binom{n}{2} \Big/ 2. \tag{16}$$

Используя тождество

$$\binom{N}{n}\binom{n}{2} = \binom{N}{2}\binom{N-2}{n-2},$$

которое представляет собой частный случай соотношения 1.2.6–(20), можно легко определить сумму в (16):

$$B_{\text{ave}} = \frac{1}{2M}\binom{N}{2}. \tag{17}$$

В упр. 37 определено стандартное отклонение для B, но суммирование в (15) выполняется сложнее. По теореме 1.2.7A получим

$$\sum_n \binom{N}{n}(M-1)^{-n} H_n = \left(1 - \frac{1}{M}\right)^{-N} (H_N - \ln M) + \epsilon,$$

$$0 < \epsilon = \sum_{n>N} \frac{1}{n}\left(1 - \frac{1}{M}\right)^{n-N} < \frac{M-1}{N+1};$$

следовательно,

$$A_{\text{ave}} = M(H_N - \ln M) + \delta, \qquad 0 < \delta < \frac{M^2}{N+1}\left(1 - \frac{1}{M}\right)^{N+1}. \tag{18}$$

(Эта формула практически бесполезна, если $M \approx N$. Более подробно асимптотическое поведение величины A_{ave} при $M = N/\alpha$ обсуждается в упр. 40.)

Рассматривая совместно (17) и (18), можно получить общее время выполнения программы M при фиксированном M и $N \to \infty$:

$$\begin{aligned} \min\ & 31N + M + 2, \\ \text{ave}\ & 1.75N^2/M + 31N - 3MH_N + 3M\ln M + 4M - 3\delta - 1.75N/M + 2, \\ \max\ & 3.50N^2 + 24.5N + 4M + 2. \end{aligned} \tag{19}$$

Заметим, что если M не слишком велико, то *среднее время выполнения сокращается в M раз*. При $M = 10$ сортировка будет осуществляться примерно в 10 раз быстрее, чем при $M = 1$. Однако максимальное время выполнения гораздо больше среднего. Таким образом, подтверждается важность выполнения условия равномерности распределения ключей, так как наихудший случай имеет место, когда все ключи попадают в один список.

Если положить $M = N$, то среднее время выполнения будет составлять примерно $34.36N$ машинных циклов. При $M = \frac{1}{2}N$ оно несколько больше, приблизительно $34.52N$ машинных циклов, а при $M = \frac{1}{10}N$ — приблизительно $48.04N$. (Заметим, что $10N$ машинных циклов MIX при этом тратятся на одну лишь команду умножения!) Дополнительные затраты на сопровождающую программу из упр. 35, которая связывает воедино все M списков, увеличивают время выполнения приблизительно до $44.99N$, $41.95N$ и $52.74N$ соответственно вариантам значения M. *Таким образом, получен метод сортировки с временем работы порядка N при условии, что ключи довольно равномерно распределены в области возможных значений.*

Пути дальнейшего совершенствования метода вставок в несколько списков обсуждаются ниже, в разделе 5.2.5.

УПРАЖНЕНИЯ

1. [*10*] Является ли алгоритм S алгоритмом "устойчивой" сортировки?

2. [*11*] Будет ли алгоритм S правильно сортировать числа, если на шаге S3 отношение $K \geq K_i$ заменить отношением $K > K_i$?

▶ **3.** [*30*] Является ли программа S самой короткой программой сортировки для машины MIX, или можно написать еще более короткую программу, которая давала бы тот же результат?

▶ **4.** [*M20*] Найдите минимальное и максимальное время выполнения программы S как функцию от N.

▶ **5.** [*M27*] Найдите производящую функцию $g_N(z) = \sum_{k\geq 0} p_{Nk} z^k$ для общего времени выполнения программы S, где p_{Nk} — вероятность того, что на выполнение программы S уйдет ровно k машинных циклов при заданной исходной случайной перестановке множества $\{1, 2, \ldots, N\}$. Вычислите также стандартное отклонение времени выполнения для данного значения N.

6. [*23*] Для метода двухпутевых вставок, проиллюстрированного в табл. 2, по-видимому, необходимо, помимо области ввода, содержащей N записей, иметь область вывода, в которой может храниться $2N + 1$ записей. Покажите, что можно выполнять двухпутевые вставки, имея в памяти как для ввода, так и для вывода пространство, достаточное для хранения всего $2N + 1$ записей.

7. [*M20*] Пусть $a_1 a_2 \ldots a_n$ — случайная перестановка множества $\{1, 2, \ldots, n\}$. Каково среднее значение величины $|a_1 - 1| + |a_2 - 2| + \cdots + |a_n - n|$? (Оно равно произведению n и среднего чистого расстояния, на которое перемещается запись в процессе сортировки.)

8. [*10*] Является ли алгоритм D алгоритмом "устойчивой" сортировки?

9. [*20*] Какие значения A и B и какое общее время работы программы D соответствуют табл. 3 и 4? Проанализируйте достоинства метода Шелла по сравнению с методом простых вставок в этом случае.

▶ **10.** [*22*] В случае, когда $K_j \geq K_{j-h}$, на шаге D3 алгоритм D предписывает выполнение множества ненужных действий. Покажите, как можно изменить программу D, чтобы избежать этих избыточных вычислений, и проанализируйте преимущества такой модификации.

11. [*M10*] Какой путь на решетке (аналогичной представленной на рис. 11) соответствует перестановке 1 2 5 3 7 4 8 6 9 11 10 12?

12. [*M20*] Докажите, что сумма вертикальных весов пути на решетке равна числу инверсий соответствующей 2-упорядоченной перестановки.

▶ **13.** [*M16*] Поясните, как нужно приписать веса на решетке *горизонтальным отрезкам* вместо вертикальных, чтобы сумма горизонтальных весов пути на решетке равнялась числу инверсий в соответствующей 2-упорядоченной перестановке.

14. [*M28*] (a) Покажите, что для сумм, определяемых равенством (2), действительно $A_{2n+1} = 2A_{2n}$. (b) Если положить $r = s$, $t = -2$, то обобщенное тождество в упр. 1.2.6–26 упрощается и приводится к виду

$$\sum_k \binom{2k+s}{k} z^k = \frac{1}{\sqrt{1-4z}} \left(\frac{1-\sqrt{1-4z}}{2z} \right)^s .$$

Проанализировав сумму $\sum_n A_{2n} z^n$, покажите, что

$$A_{2n} = n \cdot 4^{n-1}.$$

▸ **15.** [*HM33*] Пусть $g_n(z)$, $\hat{g}_n(z)$, $h_n(z)$ и $\hat{h}_n(z)$ равны $\sum z^{\text{общий вес пути}}$, где сумма берется по всем путям длиной $2n$ на решетке от (0,0) до (n,n), где веса определяются, как на рис. 11, а пути удовлетворяют некоторым ограничениям на вершины, через которые эти пути проходят. Для $h_n(z)$ нет ограничений, но для $g_n(z)$ пути не должны проходить через вершины (i,j), такие, что $i > j$; $\hat{h}_n(z)$ и $\hat{g}_n(z)$ определяются аналогично, но не допускаются также и вершины (i,i) при $0 < i < n$. Таким образом,

$$g_0(z) = 1, \qquad g_1(z) = z, \qquad g_2(z) = z^3 + z^2; \qquad \hat{g}_1(z) = z, \qquad \hat{g}_2(z) = z^3;$$

$$h_0(z) = 1, \qquad h_1(z) = z + 1, \qquad h_2(z) = z^3 + z^2 + 3z + 1;$$

$$\hat{h}_1(z) = z + 1, \qquad \hat{h}_2(z) = z^3 + z.$$

Найдите рекуррентные соотношения, определяющие эти функции; и воспользуйтесь полученными рекуррентными соотношениями для доказательства равенства

$$h_n''(1) + h_n'(1) = \frac{7n^3 + 4n^2 + 4n}{30} \binom{2n}{n}.$$

(Отсюда легко можно найти точную формулу дисперсии числа инверсий в случайной 2-упорядоченной перестановке множества $\{1, 2, \ldots, 2n\}$; она асимптотически приближается к $(\frac{7}{30} - \frac{\pi}{16})n^3$.)

16. [*M24*] Найдите формулу максимального числа инверсий в h-упорядоченной перестановке множества $\{1, 2, \ldots, n\}$. Каково максимально возможное число перезаписей при выполнении алгоритма D, если значения смещений для сортировки удовлетворяют условию делимости (5)?

17. [*M21*] Покажите, что если $N = 2^t$ и $h_s = 2^s$ при $t > s \geq 0$, то существует единственная перестановка множества $\{1, 2, \ldots, n\}$, максимизирующая число перемещений записей при выполнении алгоритма D. Найдите простой способ описания этой перестановки.

18. [*HM24*] При больших значениях N сумму (6) можно считать равной

$$\frac{1}{4}\frac{N^2}{h_{t-1}} + \frac{\sqrt{\pi}}{8}\left(\frac{N^{3/2}h_{t-1}^{1/2}}{h_{t-2}} + \cdots + \frac{N^{3/2}h_1^{1/2}}{h_0}\right).$$

Каковы действительные значения $h_{t-1}, \ldots, h_0$, минимизирующие это выражение, если N и t фиксированы, а $h_0 = 1$?

▸ **19.** [*M25*] Каково среднее значение величины A в анализе времени работы программы D, если значения смещений при сортировке удовлетворяют условию делимости (5)?

20. [*M22*] Покажите, что теорема K следует из леммы L.

21. [*M25*] Пусть h и k — взаимно простые целые положительные числа. Будем говорить, что целое число — *порождаемое*, если оно равно $xh + yk$ при некоторых неотрицательных целых x_k и y_k. Покажите, что n является порождаемым тогда и только тогда, когда $hk - h - k - n$ — не порождаемое. (Поскольку 0 — наименьшее порождаемое целое, наибольшее не порождаемое должно, таким образом, быть равно $hk - h - k$. Отсюда следует, что в любом массиве, который является одновременно и h-, и k-упорядоченным, $K_i \leq K_j$, если только $j - i \geq (h-1)(k-1)$.)

22. [*M30*] Докажите, что любое целое число $\geq 2^s(2^s - 1)$ можно представить в виде

$$a_0(2^s - 1) + a_1(2^{s+1} - 1) + a_2(2^{s+2} - 1) + \cdots,$$

где a_j — неотрицательные целые числа; но число $2^s(2^s-1)-1$ нельзя представить в таком виде. Более того, существует ровно $2^{s-1}(2^s+s-3)$ целых положительных чисел, которые нельзя представить в таком виде.

Найдите аналогичные формулы для случая, когда в этом представлении выражение 2^k-1 заменено выражением 2^k+1.

▶ **23.** [*M22*] Докажите, что если h_{s+2} и h_{s+1} — взаимно простые числа, то количество перезаписей в ходе выполнения алгоритма D при просмотре со смещением h_s есть $O(Nh_{s+2}h_{s+1}/h_s)$. (*Указание.* См. упр. 21.)

24. [*M42*] Докажите, что теорема P — наилучшая из возможных теорем в том смысле, что показатель 3/2 нельзя уменьшить.

▶ **25.** [*M22*] Сколько существует перестановок множества $\{1,2,\ldots,N\}$, являющихся одновременно 3- и 2-упорядоченными? Каково максимальное число инверсий в такой перестановке? Чему равно суммарное число инверсий во всех таких перестановках?

26. [*M35*] Может ли 3-, 5- и 7-упорядоченный массив из N элементов содержать более N инверсий? Оцените максимальное количество инверсий при большом N.

27. [*M41*] (Бьерн Пунен (Bjorn Poonen).) (a) Докажите, что существует константа c, такая, что если m из смещений h_s в алгоритме D имеют величину, меньшую $N/2$, то время выполнения будет в худшем случае равно $\Omega(N^{1+c/\sqrt{m}})$. (b) Следовательно, время выполнения алгоритма D будет в худшем случае равно $\Omega(N(\log N/\log\log N)^2)$ для всех последовательностей смещений.

28. [*15*] Какая из последовательностей смещений, указанных в табл. 6, наилучшая для программы D с точки зрения суммарного времени выполнения сортировки?

29. [*40*] Найдите для $N=1000$ и различных значений t эмпирические значения $h_{t-1},\ldots,h_1,h_0$, которые, быть может, минимизируют среднее число операций перемещения записей B_{ave} в алгоритме D.

30. [*M23*] (В. Пратт (V. Pratt).) Покажите, что если множество смещений при сортировке методом Шелла есть $\{2^p3^q \mid 2^p3^q<N\}$, то число проходов равно примерно $\frac{1}{2}(\log_2 N)(\log_3 N)$ и на каждый просмотр приходится не более $N/2$ операций перестановки записей. В действительности, если $K_{j-h}>K_j$, то на любом проходе $K_{j-3h},K_{j-2h}\le K_j<K_{j-h}\le K_{j+h},K_{j+2h}$; так что можно просто поменять местами K_{j-h} и K_j и увеличить j на $2h$, сэкономив два сравнения в алгоритме D. (*Указание.* См. упр. 25.)

▶ **31.** [*25*] Напишите MIX-программу для алгоритма сортировки, предложенного Праттом (упр. 30). Выразите время ее выполнения через параметры A, B, S, T, N, аналогичные соответствующим параметрам в программе D.

32. [*10*] Каково будет окончательное содержимое связей $L_0L_1\ldots L_{16}$, если провести до конца сортировку списка посредством вставок в табл. 8?

▶ **33.** [*25*] Как усовершенствовать программу L, чтобы время ее выполнения определялось величиной $5B$, а не $7B$, где B5B — число инверсий. Проанализируйте возможность подобного улучшения применительно к программе S.

34. [*M10*] Проверьте формулу (14).

35. [*21*] Напишите MIX-программу, которая должна выполняться после программы M и объединять все списки в единый список. Созданная программа должна устанавливать поля LINK точно так, как если бы они были заполнены программой L.

36. [*18*] Предположим, что размер байта компьютера MIX равен ста и значения шестнадцати ключей в табл. 8 равны 503000, 087000, 512000, ..., 703000. Определите время работы программ L и M с этими данными при $M=4$.

37. [*M25*] Пусть $g_n(z)$ — производящая функция для числа инверсий в случайной перестановке n элементов (ср. с формулой 5.1.1–(11)). Пусть $g_{NM}(z)$ — соответствующая производящая функция для величины B в программе M. Покажите, что

$$\sum_{N\geq 0} g_{NM}(z)\frac{M^N w^N}{N!} = \left(\sum_{n\geq 0} g_n(z)\frac{w^n}{n!}\right)^M,$$

и воспользуйтесь этой формулой для вывода дисперсии величины B.

38. [*HM23*] (Р. М. Карп (R. M. Karp).) Пусть $F(x)$ — некоторая функция распределения вероятностей, причем $F(0) = 0$ и $F(1) = 1$. Докажите, что если ключи $K_1, K_2, \ldots, K_N$ независимо выбираются случайным образом из последовательности чисел с этим распределением и $M = cN$, где c — константа, а $N \to \infty$, то время выполнения программы M есть $O(N)$, если только F — достаточно гладкая функция. (Ключ K вставляется в список j, если $\lfloor MK \rfloor = j - 1$; это случается с вероятностью $F(j/M) - F((j-1)/M)$. В тексте раздела рассматривался лишь случай $F(x) = x$, $0 \leq x \leq 1$.)

39. [*HM16*] Если программа выполняется приблизительно за $A/M + B$ машинных циклов и использует $C + M$ ячеек памяти, то при каком выборе M достигается минимальное значение произведения пространства памяти на время выполнения?

▸ **40.** [*HM24*] Найдите выражение для асимптотической величины среднего числа максимумов слева направо (формула (15)), которое возиикает при множестве вставок в список, если $M = N/\alpha$ для фиксированного α при $N \to \infty$. Проанализируйте поведение абсолютной ошибки $O(N^{-1})$, сформулировав ответ в терминах *показательной интегральной* функции $E_1(z) = \int_z^\infty e^{-t}\,dt/t$.

41. [*HM26*] (a) Докажите, что сумма первых $\binom{k}{2}$ элемеитов в (10) равна $O(\rho^{2k})$. (b) Теперь докажите теорему I.

42. [*HM43*] Проанализируйте поведение усредненного показателя в ходе выполнения сортировки Шелла, если имеется $t = 3$ смещений h, g и 1 и полагается, что $h \perp g$. Очевидно, что на первом проходе при выполнении h-сортировки получим в среднем $\frac{1}{4}N^2/h + O(N)$ перемещений записей.

a) Докажите, что второй проход, g-сортировка, даст $\frac{\sqrt{\pi}}{8}(\sqrt{h} - 1/\sqrt{h})N/g + O(hN)$ перемещений записей.

b) Докажите, что третий проход, 1-сортировка, даст $\psi(h,g)N + O(g^3h^2)$ перемещений, где

$$\psi(h,g) = \frac{1}{2}\sum_{d=1}^{g-1}\sum_j \binom{h-1}{j}\left(\frac{d}{g}\right)^j\left(1-\frac{d}{g}\right)^{h-1-j}\left|j - \left\lfloor\frac{hd}{g}\right\rfloor\right|.$$

▸ **43.** [*25*] В упр. 33 для ускоренного выполнения алгоритма S используется прием, который делает ненужной проверку "$i > 0$" на шаге S4. Этот фокус не проходит в алгоритме D. Покажите, что, тем не менее, существует возможность избавиться от проверки "$i > 0$" на шаге D5, повысив тем самым скорость выполнения внутреннего цикла сортировки Шелла.

44. [*M25*] Если $\pi = a_1 \ldots a_n$ и $\pi' = a'_1 \ldots a'_n$ — суть перестановки множества $\{1, \ldots, n\}$, будем говорить, что $\pi \leq \pi'$, если i-й наибольший элемент в $\{a_1, \ldots, a_j\}$ меньше или равен i-му наибольшему элементу в $\{a'_1, \ldots, a'_j\}$ при $1 \leq i \leq j \leq n$. (Другими словами, $\pi \leq \pi'$, если для всех j сортировка посредством простых вставок π является покомпонентно меньшей или равной сортировке посредством прямых вставок π' после того, как вставлены первые j элементов.)

a) Если π находится "выше" π' в смысле, определенном в упр. 5.1.1–12, следует ли из этого, что $\pi \leq \pi'$?

b) Если $\pi \leq \pi'$, следует ли из этого, что $\pi^R \geq \pi'^R$?

c) Если $\pi \leq \pi'$, следует ли из этого, что π расположена выше π'?

5.2.2. Обменная сортировка

Мы подошли ко второму из семейств алгоритмов сортировки, упомянутых в самом начале раздела 5.2, — к методам обменов или транспозиций, предусматривающих систематический обмен местами между элементами пар, в которых нарушается упорядоченность, до тех пор, пока таких пар не останется.

Реализацию метода простых вставок (алгоритм 5.2.1S) можно рассматривать как обменную сортировку: берем новую запись R_j и, по существу, меняем местами с соседями слева до тех пор, пока она не займет нужное место. Следовательно, классификацию методов сортировки по таким семействам, как вставки, обмены, выбор и т. д., нельзя считать слишком строгой. В этом разделе рассматриваются четыре типа методов сортировки, для которых обмен является основной операцией: *обменную сортировку с выбором* (метод пузырька), *обменную сортировку со слиянием* (параллельную сортировку Бэтчера), *обменную сортировку с разделением* (быструю сортировку Хоара) и *поразрядную обменную сортировку*.

Метод пузырька. Пожалуй, наиболее очевидный способ обменной сортировки — сравнить K_1 с K_2, меняя местами R_1 и R_2, если их ключи расположены не в нужном порядке, и затем проделать то же самое с R_2 и R_3, R_3 и R_4 и т. д. При выполнении этой последовательности операций записи с большими ключами будут продвигаться вправо; и действительно, все это закончится тем, что запись с наибольшим ключом займет положение R_N. При многократном выполнении данного процесса соответствующие записи попадут в позиции R_{N-1}, R_{N-2} и т. д., так что, в конце концов, все записи будут упорядочены. На рис. 14 показано использование этого метода сортировки для шестнадцати ключей: 503 087 512 ... 703. Последовательность чисел удобно представлять не горизонтально, а вертикально, чтобы запись R_N была в самом верху, а R_1 — в самом низу. Метод назван "методом пузырька", потому что большие элементы, подобно пузырькам, "всплывают" на соответствующую позицию в противоположность "методу погружения" (т. е. методу простых вставок), в котором элементы погружаются на соответствующий уровень. Метод пузырька известен и под более прозаическими именами, такими как "обменная сортировка с выбором" и "метод распространения".

Нетрудно видеть, что после каждого просмотра последовательности все записи, расположенные выше самой последней, которая участвовала в обмене, и сама эта запись должны занять свои окончательные позиции, так что их не нужно проверять при последующих просмотрах. На рис. 14 процесс сортировки с этой точки зрения отмечен подчеркиванием последнего элемента, занявшего верную позицию. Обратите внимание, например, на то, что после четвертого прохода видно, что еще пять записей сразу заняли свои окончательные позиции. При последнем просмотре перемещения записей вообще не было. Теперь, понаблюдав за процессом, мы готовы сформулировать алгоритм.

Алгоритм B (*Метод пузырька*). Записи $R_1, \ldots, R_N$ перекомпоновываются в пределах того же пространства памяти; после завершения сортировки их ключи будут упорядочены так: $K_1 \le \cdots \le K_N$ (рис. 15).

B1. [Начальная установка BOUND.] Присвоить BOUND $\leftarrow N$. (BOUND — индекс самого верхнего элемента, о котором еще не известно, занял ли он свою окончательную

Проход 1	Проход 2	Проход 3	Проход 4	Проход 5	Проход 6	Проход 7	Проход 8	Проход 9
703	908	908	908	908	908	908	908	908
765	703	897	897	897	897	897	897	897
677	765	703	765	765	765	765	765	765
612	677	765	703	703	703	703	703	703
509	612	677	677	677	677	677	677	677
154	509	612	653	653	653	653	653	653
426	154	509	612	612	612	612	612	612
653	426	154	509	512	512	512	512	512
275	653	426	154	509	509	509	509	509
897	275	653	426	154	503	503	503	503
170	897	275	512	426	154	426	426	426
908	170	512	275	503	426	154	275	275
061	512	170	503	275	275	275	154	170
512	061	503	170	170	170	170	170	154
087	503	061	087	087	087	087	087	087
503	087	087	061	061	061	061	061	061

Рис. 14. Процесс сортировки методом пузырька.

позицию; таким образом, можно отметить, что в начале сортировки еще ничего не известно о порядке размещения записей.)

B2. [Цикл по j.] Присвоить $t \leftarrow 0$. Выполнить шаг B3 при $j = 1, 2, \ldots, \texttt{BOUND} - 1$. Затем перейти к шагу B4. (Если $\texttt{BOUND} = 1$, то сразу перейти к B4.)

B3. [Сравнение/обмен $R_j : R_{j+1}$.] Если $K_j > K_{j+1}$, то поменять местами $R_j \leftrightarrow R_{j+1}$ и установить $t \leftarrow j$.

B4. [Были ли обмены?] Если $t = 0$, завершить выполнение процедуры. В противном случае присвоить $\texttt{BOUND} \leftarrow t$ и возвратиться к шагу B2. ▮

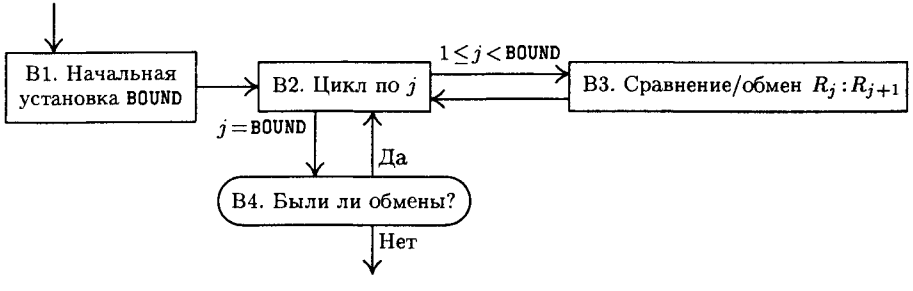


Рис. 15. Блок-схема сортировки методом пузырька.

Программа B (*Метод пузырька*). Как и в предыдущих MIX-программах этой главы, полагаем, что сортируемые записи находятся в ячейках от INPUT+1 до INPUT+N. rI1 $\equiv t$; rI2 $\equiv j$.

```
01  START ENT1 N              1     B1. Инициализировать BOUND. t ← N.
02  1H    ST1  BOUND(1:2)     A     BOUND ← t.
03        ENT2 1              A     B2. Цикл по j. j ← 1.
04        ENT1 0              A     t ← 0.
05        JMP  BOUND          A     Выход, если j ≥ BOUND.
06  3H    LDA  INPUT,2        C     B3. Сравнение и/или обмен R_j : R_{j+1}.
07        CMPA INPUT+1,2      C
08        JLE  2F             C     Нет обмена, если K_j ≤ K_{j+1}.
09        LDX  INPUT+1,2      B     R_{j+1}
10        STX  INPUT,2        B       → R_j.
11        STA  INPUT+1,2      B     (Прежнее R_j) → R_{j+1}.
12        ENT1 0,2            B     t ← j.
13  2H    INC2 1              C     j ← j + 1.
14  BOUND ENTX -*,2           A+C   rX ← j − BOUND.  [Модифицируемая команда]
15        JXN  3B             A+C   Выполнять шаг B3 для r 1 ≤ j < BOUND.
16  4H    J1P  1B             A     B4. Нет обменов? Перейти к шагу B2,
                                    если t > 0.  ▮
```

Анализ метода пузырька. Очень полезно исследовать время работы алгоритма B. Оно определяется тремя параметрами: числом проходов A, числом обменов B и числом сравнений C. Если исходные ключи различны и расположены в случайном порядке, можно предположить, что они образуют случайную перестановку множества $\{1, 2, \ldots, n\}$. Понятие *таблица инверсий* (раздел 5.1.1) приводит к простому способу описания эффекта, достигаемого на каждом проходе при сортировке методом пузырька.

Теорема I. *Пусть $a_1\, a_2 \ldots a_n$ — перестановка множества $\{1, 2, \ldots, n\}$, а $b_1\, b_2 \ldots b_n$ — соответствующая таблица инверсий. Если в результате очередного прохода при сортировке методом пузырька (алгоритм B) перестановка $a_1\, a_2 \ldots a_n$ преобразуется в $a'_1\, a'_2 \ldots a'_n$, то соответствующая таблица инверсий $b'_1\, b'_2 \ldots b'_n$ получается из $b_1\, b_2 \ldots b_n$ в результате уменьшения на единицу каждого ненулевого элемента.*

Доказательство. Если перед a_i имеется больший элемент, то a_i поменяется местами с наибольшим из предшествующих элементов, так что b_{a_i} уменьшится на единицу. С другой стороны, если перед a_i нет элемента, большего a_i, то a_i никогда не поменяется местами с большим элементом, так что b_{a_i} останется равным нулю. ▮

Итак, можно разобраться в том, что происходит в процессе сортировки методом пузырька, изучая последовательность таблиц инверсий между проходами. Вот как выглядят, например, таблицы инверсий для рис. 14.

$$\begin{array}{ll} & 3\ 1\ 8\ 3\ 4\ 5\ 0\ 4\ 0\ 3\ 2\ 2\ 3\ 2\ 1\ 0 \\ \text{Проход 1} & \\ & 2\ 0\ 7\ 2\ 3\ 4\ 0\ 3\ 0\ 2\ 1\ 1\ 2\ 1\ 0\ 0 \\ \text{Проход 2} & \\ & 1\ 0\ 6\ 1\ 2\ 3\ 0\ 2\ 0\ 1\ 0\ 0\ 1\ 0\ 0\ 0 \\ \text{Проход 3} & \\ & 0\ 0\ 5\ 0\ 1\ 2\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0 \end{array} \qquad (1)$$

Поэтому, если $b_1\, b_2 \ldots b_n$ — таблица инверсий исходной перестановки, то должны выполняться равенства

$$A = 1 + \max(b_1, b_2, \ldots, b_n), \tag{2}$$

$$B = b_1 + b_2 + \cdots + b_n, \tag{3}$$

$$C = c_1 + c_2 + \cdots + c_A, \tag{4}$$

где c_j — значение BOUND − 1 перед началом j-го прохода. Используя таблицы инверсий, запишем

$$c_j = \max\{b_i + i \mid b_i \geq j - 1\} - j \tag{5}$$

(см. упр. 5). Следовательно, в примере (1) $A = 9$, $B = 41$, $C = 15+14+13+12+7+5+4+3+2 = 75$. Общее время сортировки на машине MIX для случая, показанного на рис. 14, равно $960u$.

Распределение величины B (суммарного числа инверсий в случайной перестановке) нам уже хорошо известно; значит, остается проанализировать величины A и C.

Вероятность того, что $A \leq k$, равна произведению $1/n!$ и числа таблиц инверсий, не содержащих компонент $\geq k$, а именно — $k^{n-k}k!$ при $1 \leq k \leq n$. Следовательно, вероятность того, что потребуется ровно k проходов, равна

$$A_k = \frac{1}{n!}\left(k^{n-k}k! - (k-1)^{n-k+1}(k-1)!\right). \tag{6}$$

Теперь можно вычислить среднее значение $\sum kA_k$. Выполнив суммирование по частям, получаем

$$A_{\text{ave}} = n + 1 - \sum_{k=0}^{n} \frac{k^{n-k}k!}{n!} = n + 1 - P(n), \tag{7}$$

где $P(n)$ — функция, асимптотическое поведение которой, как следует из соотношения 1.2.11.3–(24), описывается формулой $\sqrt{\pi n/2} - \frac{2}{3} + O\left(1/\sqrt{n}\right)$. Формула (7) была представлена без доказательства в работе E. H. Friend, *JACM* **3** (1956), 150; доказательство в своей докторской диссертации привел Говард Б. Демут (Howard B. Demuth) [Ph. D. Thesis (Stanford University, October, 1956), 64–68]. Стандартное отклонение величины A представлено в упр. 7.

Проанализировать суммарное число сравнений C несколько сложнее, поэтому рассмотрим только среднее значение C_{ave}. Пусть $f_j(k)$ — число таких таблиц инверсий $b_1 \ldots b_n$ (n фиксировано), что при $1 \leq i \leq n$ либо $b_i < j-1$, либо $b_i + i - j \leq k$; тогда

$$f_j(k) = (j+k)!\,(j-1)^{n-j-k} \qquad \text{для } 0 \leq k \leq n-j \tag{8}$$

(см. упр. 8). Среднее значение c_j в (5) равно $\left(\sum k\left(f_j(k) - f_j(k-1)\right)\right)/n!$. Суммируя по частям, а затем по j, получаем формулу

$$C_{\text{ave}} = \binom{n+1}{2} - \frac{1}{n!}\sum_{\substack{1\leq j\leq n\\ 0\leq k\leq n-j}} f_j(k) = \binom{n+1}{2} - \frac{1}{n!}\sum_{0\leq r<s\leq n} s!\,r^{n-s}. \tag{9}$$

Определить асимптотическое значение здесь не так просто, и мы вернемся к нему в конце этого раздела. Подводя итог анализу метода пузырька, запишем формулы, выведенные выше (а также ниже), следующим образом:

$$A = \bigl(\min 1,\ \text{ave}\ N - \sqrt{\pi N/2} + O(1),\ \max N\bigr); \tag{10}$$

$$B = \bigl(\min 0,\ \text{ave}\ \tfrac{1}{4}(N^2 - N),\ \max \tfrac{1}{2}(N^2 - N)\bigr); \tag{11}$$

$$C = \bigl(\min N - 1,\ \text{ave}\ \tfrac{1}{2}\bigl(N^2 - N\ln N - (\gamma + \ln 2 - 1)N\bigr) + O\bigl(\sqrt{N}\bigr),$$
$$\max \tfrac{1}{2}(N^2 - N)\bigr). \tag{12}$$

Во всех случаях минимум достигается, когда исходная последовательность записей уже упорядочена, а максимум — когда записи расположены в обратном порядке. Таким образом, время работы для машины MIX равно $8A + 7B + 8C + 1 = \bigl(\min 8N + 1,\ \text{ave}\ 5.75N^2 + O(N\log N),\ \max 7.5N^2 + 0.5N + 1\bigr)$.

Модификация метода пузырька. Мы потратили много усилий на анализ метода пузырька, и, хотя способы, применявшиеся при вычислениях, поучительны, результаты разочаровывают, поскольку они говорят о том, что метод пузырька вовсе не так уж хорош. По сравнению с простыми вставками (алгоритм 5.2.1S) метод пузырька описывается более сложной программой и требует примерно в 2 раза больше времени!

Можно предложить несколько путей улучшения метода пузырька. Например, на рис. 14 первое сравнение на проходе 4 лишнее, так же как и два первых сравнения на проходе 5 и три первых на проходах 6 и 7. Заметим, кроме того, что за один проход элемент не может переместиться более чем на одну позицию влево; так что если наименьший элемент вначале был крайним справа, то придется выполнить максимальное число сравнений. Это наводит на мысль о "шейкер-сортировке", когда последовательность записей просматривается попеременно в обоих направлениях (рис. 16). При таком подходе среднее число сравнений несколько сокращается. К. Э. Айверсон [см. K. E. Iverson, *A Programming Language* (Wiley, 1962), 218–219] сделал интересное в этом отношении наблюдение: если j — такой индекс, что R_j и R_{j+1} не меняются местами на двух последовательных проходах в противоположных направлениях, то записи R_j и R_{j+1} должны занимать свои окончательные позиции и их можно исключить из последующих сравнений. Например, просматривая перестановку 4 3 2 1 8 6 9 7 5 слева направо, получаем 3 2 1 4 6 8 7 5 9; записи R_4 и R_5 не поменялись местами. При просмотре последней перестановки справа налево R_4 все еще меньше записи R_5 (новой записи). Следовательно, можно сразу же сделать вывод о том, что записи R_4 и R_5 могут и не участвовать ни в одном из последующих сравнений.

Однако ни одно из этих усовершенствований не приводит к лучшему варианту алгоритма, чем алгоритм сортировки методом простых вставок, а мы уже знаем, что даже он не годится при больших N. Другая идея состоит в том, чтобы избегать большинства обменов. Так как бóльшая часть элементов во время обменов просто сдвигается на один шаг влево, можно было бы достичь того же эффекта, рассматривая массив иначе: сместив базу индексирования! Но полученный алгоритм не превосходит метода *простого выбора* (алгоритм 5.2.3S), о котором речь пойдет несколько ниже.

Короче говоря, метод пузырька, кажется, не обладает никакими достоинствами, за которые его можно было бы порекомендовать, если не считать легко запоминающегося названия и интересных теоретических задач, к которым он приводит.

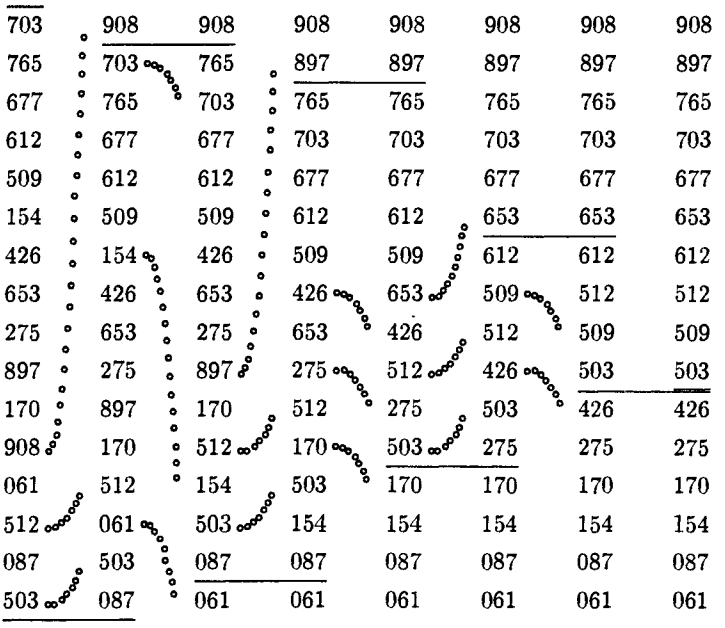


Рис. 16. Шейкер-сортировка.

Параллельная сортировка Бэтчера. Чтобы получить алгоритм обменной сортировки, время работы которого имеет порядок, меньший N^2, необходимо подобрать для сравнений пары *несоседних* ключей (K_i, K_j); иначе придется выполнить столько операций обмена записей, сколько инверсий имеется в исходной перестановке. Среднее число инверсий равно $\frac{1}{4}(N^2 - N)$. В 1964 году К. Э. Бэтчер [см. K. E. Batcher *Proc. AFIPS Spring Joint Computer Conference* **32** (1968), 307–314] открыл интересный способ программирования последовательности сравнений, предназначенной для поиска возможных обменов. Его метод далеко не очевиден. В самом деле, обосновать его справедливость весьма сложно, поскольку выполняется относительно мало сравнений. Рассмотрим два доказательства: одно в этом разделе, а другое — в разделе 5.3.4.

Схема сортировки Бэтчера несколько напоминает сортировку Шелла, но сравнения выполняются по-новому, а потому цепочки операций обмена записей не возникает. В качестве примера сравним табл. 1 и 5.2.1–3. Сортировка Бэтчера действует, как 8-, 4-, 2- и 1-сортировка, но сравнения не перекрываются. Поскольку в алгоритме Бэтчера, по существу, происходит слияние пар рассортированных подпоследовательностей, его можно назвать обменной сортировкой со слиянием.

Алгоритм М (*Обменная сортировка со слиянием*). Записи $R_1, \ldots, R_N$ перекомпоновываются в пределах того же пространства в памяти. После завершения сортировки их ключи будут упорядочены: $K_1 \le \cdots \le K_N$. Предполагается, что $N \ge 2$ (рис. 17).

М1. [Начальная установка p.] Установить $p \leftarrow 2^{t-1}$, где $t = \lceil \lg N \rceil$ — наименьшее целое число, такое, что $2^t \ge N$. (Шаги М2–М5 будут выполняться с $p = 2^{t-1}$, $2^{t-2}, \ldots, 1$.)

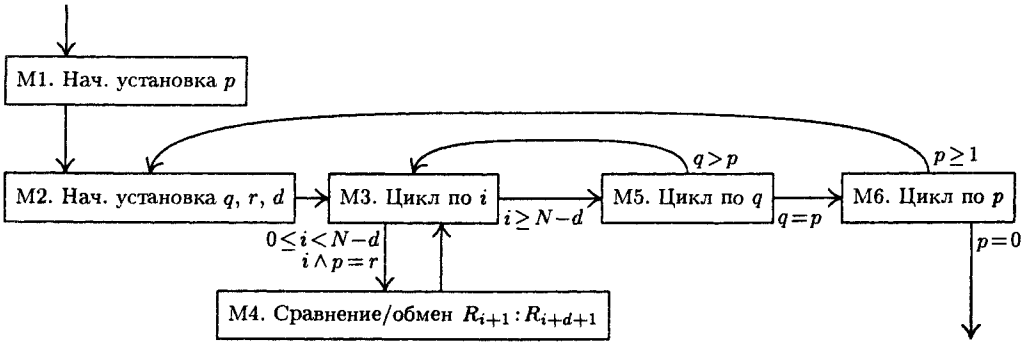


Рис. 17. Алгоритм М.

M2. [Начальная установка q, r, d.] Установить $q \leftarrow 2^{t-1}$, $r \leftarrow 0$, $d \leftarrow p$.

M3. [Цикл по i.] Для всех i, таких, что $0 \leq i < N - d$ и $i \wedge p = r$, выполнить шаг M4. Затем перейти к шагу M5. (Здесь через $i \wedge p$ обозначена операция "поразрядное логическое И" над представлениями целых чисел i и p; все биты результата равны 0, кроме тех битов, для которых в соответствующих разрядах i и p находятся 1. Так, $13 \wedge 21 = (1101)_2 \wedge (10101)_2 = (00101)_2 = 5$. К этому моменту d — нечетное кратное p (т. е. частное от деления d на p нечетно), а p — степень двойки, так что $i \wedge p \neq (i + d) \wedge p$. Отсюда следует, что шаг M4 можно выполнять при всех нужных значениях i в любом порядке или даже одновременно.)

M4. [Сравнение/обмен $R_{i+1}:R_{i+d+1}$.] Если $K_{i+1} > K_{i+d+1}$, поменять местами записи $R_{i+1} \leftrightarrow R_{i+d+1}$.

M5. [Цикл по q.] Если $q \neq p$, установить $d \leftarrow q - p$, $q \leftarrow q/2$, $r \leftarrow p$ и возвратиться к шагу M3.

M6. [Цикл по p.] (К этому моменту перестановка $K_1\, K_2 \ldots K_N$ будет p-упорядочена.) Установить $p \leftarrow \lfloor p/2 \rfloor$. Если $p > 0$, возвратиться к шагу M2. ▮

В табл. 1 этот метод проиллюстрирован при $N = 16$. Обратите внимание на то, что, по существу, алгоритм сортирует N элементов путем независимой сортировки подмассивов $R_1, R_3, R_5, \ldots$ и $R_2, R_4, R_6, \ldots$, после чего выполняются шаги M2–M5 при $p = 1$ для слияния двух рассортированных последовательностей.

Чтобы доказать, что магическая последовательность сравнений и/или обменов, описанная в алгоритме M, действительно позволяет рассортировать любую последовательность $R_1\, R_2 \ldots R_N$, необходимо показать только, что в результате выполнения шагов M2–M5 при $p = 1$ будет слита любая 2-упорядоченная последовательность $R_1\, R_2 \ldots R_N$. С этой целью можно воспользоваться методом решеточных диаграмм из раздела 5.2.1 (см. рис. 11 на с. 106); каждая 2-упорядоченная перестановка множества $\{1, 2, \ldots, N\}$ соответствует на решетке единственному пути из вершины (0,0) к $(\lceil N/2 \rceil, \lfloor N/2 \rfloor)$. На рис. 18, (а) показан пример при $N = 16$, соответствующий перестановке 1 3 2 4 10 5 11 6 13 7 14 8 15 9 16 12. При $p = 1$, $q = 2^{t-1}$, $r = 0$, $d = 1$ на шаге M3 выполняется сравнение (и, возможно, обмен) записей $R_1 : R_2$, $R_3 : R_4$ и т. д. Этой операции соответствует простое преобразование пути на решетке:

Таблица 1

ОБМЕННАЯ СОРТИРОВКА СО СЛИЯНИЕМ (МЕТОД БЭТЧЕРА)

p q r d																
	503	087	512	061	908	170	897	275	653	426	154	509	612	677	765	703
8 8 0 8																
	503	087	154	061	612	170	765	275	653	426	512	509	908	677	897	703
4 8 0 4																
	503	087	154	061	612	170	765	275	653	426	512	509	908	677	897	703
4 4 4 4																
	503	087	154	061	612	170	512	275	653	426	765	509	908	677	897	703
2 8 0 2																
	154	061	503	087	512	170	612	275	653	426	765	509	897	677	908	703
2 4 2 6																
	154	061	503	087	512	170	612	275	653	426	765	509	897	677	908	703
2 2 2 2																
	154	061	503	087	512	170	612	275	653	426	765	509	897	677	908	703
1 8 0 1																
	061	154	087	503	170	512	275	612	426	653	509	765	677	897	703	908
1 4 1 7																
	061	154	087	503	170	512	275	612	426	653	509	765	677	897	703	908
1 2 1 3																
	061	154	087	275	170	426	503	509	512	653	612	703	677	897	765	908
1 1 1 1																
	061	087	154	170	275	426	503	509	512	612	653	677	703	765	897	908

"перегиб" относительно диагонали при необходимости, чтобы путь нигде не проходил выше диагонали (рис. 18, (b) и доказательство в упр. 10). На следующей итерации шага M3 имеем $p = r = 1$ и $d = 2^{t-1} - 1, 2^{t-2} - 1, \ldots, 1$. В результате произойдет сравнение и/или обмен записей $R_2 : R_{2+d}$, $R_4 : R_{4+d}$ и т. д. Опять же, имеется простая геометрическая интерпретация: путь "перегибается" относительно прямой, расположенной на $\frac{1}{2}(d+1)$ единиц ниже диагонали (рис. 18, (c) и (d)). В конце концов, получаем путь, изображенный на рис. 18, (e), который соответствует полностью рассортированной перестановке. На этом "геометрическое доказательство" справедливости алгоритма Бэтчера завершается (данный алгоритм можно было бы назвать сортировкой посредством перегибов!).

MIX-программа для алгоритма M приведена в упр. 12. К сожалению, количество вспомогательных операций, необходимых для управления последовательностью сравнений, весьма велико, так что программа менее эффективна, чем другие рассмотренные выше методы. Однако она обладает одним важным компенсирующим качеством: все сравнения и/или обмены, определяемые данной итерацией шага M3, можно выполнять *одновременно* на компьютерах или в сетях, которые реализуют параллельные вычисления. С такими параллельными операциями сортировка осу-

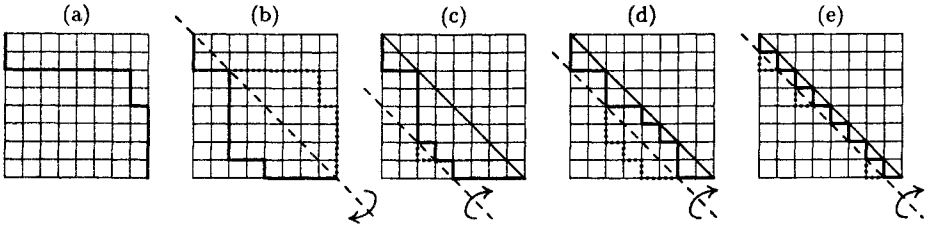


Рис. 18. Геометрическая интерпретация метода Бэтчера, $N = 16$.

ществляется за $\frac{1}{2}\lceil \lg N \rceil (\lceil \lg N \rceil + 1)$ шагов, и это один из самых быстрых общих методов среди всех известных. Например, *1 024 элемента можно рассортировать методом Бэтчера всего за 55 параллельных шагов.* Его ближайшим соперником является метод Пратта (см. упр. 5.2.1–30), который затрачивает 40 или 73 шага в зависимости от того, как считать: если допускать перекрытие сравнений до тех пор, пока не потребуется выполнять перекрывающиеся обмены, то для сортировки 1 024 элементов методом Пратта требуется всего 40 циклов сравнения и/или обмена. Дальнейшие пояснения приводятся в разделе 5.3.4.

Быстрая сортировка. В методе Бэтчера последовательность сравнений предопределена: каждый раз сравниваются одни и те же пары ключей независимо от информации о сортируемой последовательности, которую могут предоставить уже выполненные операции сравнения. Это утверждение в большой мере справедливо и применительно к методу пузырька, хотя алгоритм B использует в ограниченной степени полученные сведения, чтобы сократить объем работы в крайней справа части последовательности. Обратимся теперь к совсем иной стратегии, при которой для определения, какие ключи сравнивать следующими, используется результат каждого сравнения. Такая стратегия не годится для параллельных вычислений, но она может оказаться плодотворной для обычных компьютеров с последовательным выполнением операций.

Основная идея, на которой базируется этот метод, состоит в том, чтобы взять одну из записей, скажем R_1, и передвинуть ее на то место, которое она должна занять после сортировки, скажем в позицию s. В поиске этой окончательной позиции придется несколько перекомпоновать и остальные записи таким образом, чтобы слева от позиции s не оказалось ни одной записи с бóльшим ключом, а справа — с меньшим. В результате последовательность окажется разбитой таким образом, что исходная задача сортировки всего массива записей будет сведена к задачам независимой сортировки пары подмассивов $R_1 \ldots R_{s-1}$ и $R_{s+1} \ldots R_N$ меньшей длины. Далее, тот же метод применяется и к полученным подмассивам до тех пор, пока не будет завершена сортировка всей последовательности. Существует несколько способов разбиения всей последовательности на подмассивы. Рассмотрим следующую процедуру, предложенную Седгевиком, которая, на наш взгляд, является лучшей из имеющихся на сегодняшний день, в основном, вследствие своей "прозрачности" и простоты анализа алгоритма. Итак, пусть имеются указатели i и j, причем вначале $i = 2$ и $j = N$. Предположив, что запись R_i должна после разделения принадлежать левому подмассиву (это легко определить, сравнив K_i с K_1), увеличим i на 1 и продолжим далее до тех пор, пока не обнаружим такую запись R_i, которая

принадлежит правому подмассиву. Аналогично будем уменьшать j на 1 до тех пор, пока не обнаружим такую запись R_j, которая принадлежит левому подмассиву. Если $i < j$, поменяем местами R_i с R_j; затем повторим аналогичную процедуру со следующей записью, "сжигая свечку с обоих концов", пока не станет $i \geq j$. Завершается процесс разделения массива обменом записей R_j с R_1. Посмотрим, например, что произойдет с нашей последовательностью из шестнадцати чисел.

		i ↓														j ↓
Исходный массив:	[503	**087**	512	061	908	170	897	275	653	426	154	509	612	677	765	**703**]
1-й обмен:	503	087	**512**	061	908	170	897	275	653	426	**154**	509	612	677	765	703
2-й обмен:	503	087	154	061	**908**	170	897	275	653	**426**	512	509	612	677	765	703
3-й обмен:	503	087	154	061	426	170	**897**	**275**	653	908	512	509	612	677	765	703
Переключение указателей:	503	087	154	061	426	170	**275**	**897**	653	908	512	509	612	677	765	703
Разделенный массив:	[275	087	154	061	426	170]	**503**	[**897**	653	908	512	509	612	677	765	703]
							↑ j	↑ i								

(Для удобства анализа положения i и j ключи K_i и K_j выделены в таблице полужирным шрифтом.)

В табл. 2 показано, как выбранная нами для примеров последовательность полностью сортируется при помощи этого метода за 11 итераций. В скобки заключены подмассивы, которые еще предстоит рассортировать; в программе эти подмассивы можно представлять двумя переменными двоичными (l, r) (границы рассматриваемого в данный момент массива) и стеком дополнительных пар (l_k, r_k). Каждый раз, когда массив разбивается, помещаем в стек *больший* из подмассивов и начинаем обрабатывать оставшийся; и так до тех пор, пока не будут получены тривиально короткие массивы. Как показано в упр. 20, такая процедура гарантирует, что в стеке никогда не будет находиться более $\lg N$ элементов.

Только что описанную процедуру сортировки можно назвать *обменной сортировкой с разделением*. Ее идея принадлежит Ч. Э. Р. Хоару, интереснейшая статья которого [C. A. R. Hoare, *Comp. J.* **5** (1962), 10–15] — одно из наиболее исчерпывающих из когда-либо опубликованных сообщений об этом методе. Хоар окрестил свой метод "quicksort" ("быстрая сортировка"), и это название вполне соответствует действительности, так как внутренний цикл вычислений при реализации на любом из современных компьютеров оказывается очень быстрым. При всех сравнениях, выполняемых на текущей итерации, используется один и тот же ключ, так что соответствующее значение можно держать в регистре. Обмен тактами между сравнениями выполняется только в отношении единственного индекса. Более того, количество перезаписей данных довольно мало́ — обработка табл. 2, например, требует всего лишь 17 операций перезаписи данных.

Вспомогательные операции (требуемые для управления стеком и переменными i, j) не сложны, но все же из-за них процедура быстрой сортировки посредством разделений пригодна, в основном, при больших значениях N. Поэтому в следующем алгоритме используется несколько измененная стратегия обработки коротких подмассивов.

Таблица 2

БЫСТРАЯ СОРТИРОВКА

																(l,r)	Стек
⟦503	087	512	061	908	170	897	275	653	426	154	509	612	677	765	703⟧	(1,16)	–
⟦275	087	154	061	426	170⟧	503	[897	653	908	512	509	612	677	765	703]	(1,6)	(8,16)
⟦170	087	154	061⟧	275	426	503	[897	653	908	512	509	612	677	765	703]	(1,4)	(8,16)
⟦061	087	154⟧	170	275	426	503	[897	653	908	512	509	612	677	765	703]	(1,3)	(8,16)
061	⟦087	154⟧	170	275	426	503	[897	653	908	512	509	612	677	765	703]	(2,3)	(8,16)
061	087	154	170	275	426	503	⟦897	653	908	512	509	612	677	765	703⟧	(8,16)	–
061	087	154	170	275	426	503	⟦765	653	703	512	509	612	677⟧	897	908	(8,14)	–
061	087	154	170	275	426	503	⟦677	653	703	512	509	612⟧	765	897	908	(8,13)	–
061	087	154	170	275	426	503	⟦509	653	612	512⟧	677	703	765	897	908	(8,11)	–
061	087	154	170	275	426	503	509	⟦653	612	512⟧	677	703	765	897	908	(9,11)	–
061	087	154	170	275	426	503	509	⟦512	612⟧	653	677	703	765	897	908	(9,10)	–
061	087	154	170	275	426	503	509	512	612	653	677	703	765	897	908	–	–

Алгоритм Q (*Быстрая сортировка*). Записи $R_1, \ldots, R_N$ перекомпоновываются в пределах того же пространства памяти. По завершении сортировки их ключи будут упорядочены: $K_1 \le \cdots \le K_N$. Нужен вспомогательный стек для хранения не более чем $\lfloor \lg N \rfloor$ элементов. Этот алгоритм (рис. 19) соответствует описанной выше процедуре быстрой сортировки посредством разделений с небольшими изменениями с целью повышения эффективности.

a) Предполагается наличие искусственных ключей $K_0 = -\infty$ и $K_{N+1} = +\infty$, таких, что

$$K_0 \le K_i \le K_{N+1} \qquad \text{для } 1 \le i \le N. \tag{13}$$

(Равенство допускается.)

b) Подмассивы, состоящие из M и менее элементов, являются нерассортированными слева до самого конца выполнения процедуры. Затем выполняется единственный проход сортировки методом простых вставок, где $M \ge 1$ — параметр, который выбирается, как описано ниже. (Эта идея принадлежит Р. Седгевику и позволяет сэкономить на вспомогательных операциях, которые необходимы, если непосредственно использовать метод прямых вставок по отношению к коротким подмассивам.)

c) Записи с одинаковыми ключами меняются местами, хотя это не является строго необходимым. (Эта идея, принадлежащая Р. К. Синглтону (R. C. Singleton), способствует разделению подмассивов почти пополам, если имеются равные ключи; см. упр. 18.)

Q1. [Начальная установка.] Если $N \le M$, перейти к шагу Q9. В противном случае очистить стек и установить $l \leftarrow 1$, $r \leftarrow N$.

Q2. [Начать новую итерацию.] (Необходимо рассортировать подмассив $R_l \ldots R_r$. Из самого существа алгоритма вытекает, что $r \ge l + M$ и $K_{l-1} \le K_i \le K_{r+1}$ при $l \le i \le r$.) Установить $i \leftarrow l$, $j \leftarrow r + 1$ и $K \leftarrow K_l$. (Обсуждение наилучшего выбора K приведено ниже.)

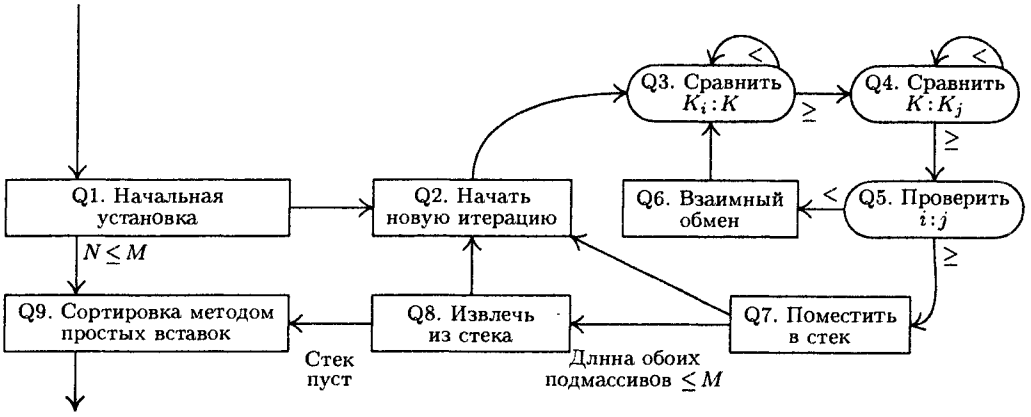


Рис. 19. Обменная сортировка с разделением (быстрая сортировка).

Q3. [Сравнить $K_i:K$.] (В этот момент массив перекомпонован таким образом, что

$$K_k \le K \quad \text{для } l-1 \le k \le i, \qquad K \le K_k \quad \text{для } j \le k \le r+1 \tag{14}$$

и $l \le i < j$.) Увеличить i на 1; затем, если $K_i < K$, повторить этот шаг. (Как только $K_j \ge K$, итерацию нужно прекратить, сохранив $i \le j$.)

Q4. [Сравнить $K:K_j$.] Уменьшить j на 1; затем, если $K < Kj$, повторить этот шаг. (Как только $K \ge K_{i-1}$, итерацию нужно прекратить, сохранив $j \ge i-1$.)

Q5. [Проверить $i:j$.] (В этот момент соблюдается условие (14), кроме случая, когда $k = i$ и $k = j$; также $K_i \ge K \ge K_j$ и $r \ge j \ge i-1 \ge l$.) Если $j \le i$, переслать $R_l \leftrightarrow R_j$ и перейти к шагу Q7.

Q6. [Взаимный обмен.] Выполнить взаимный обмен $R_i \leftrightarrow R_j$ и вернуться к шагу Q3.

Q7. [Поместить в стек.] (Теперь подмассив $R_l \ldots R_j \ldots R_r$ разделен так, что $K_k \le K_j$ при $l-1 \le k \le j$ и $K_j \le K_k$ при $j \le k \le r+1$.) Если $r-j \ge j-l > M$, то поместить в стек $(j+1, r)$, установить $r \leftarrow j-1$ и перейти к шагу Q2. Если $j-l > r-j > M$, то поместить в стек $(l, j-1)$, установить $l \leftarrow j+1$ и перейти к шагу Q2. (Каждый элемент в стеке — пара (a, b) — это запрос на сортировку подмассива $R_a \ldots R_b$, которую нужно будет выполнить позже.) В противном случае, если $r-j > M \ge j-l$, установить $l \leftarrow j+1$ и перейти к шагу Q2 или, если $j-l > M \ge r-j$, установить $r \leftarrow j-1$ и перейти к шагу Q2.

Q8. [Извлечь из стека.] Если стек не пуст, извлечь верхний элемент стека (l', r'), установить $l \leftarrow l'$, $r \leftarrow r'$ и возвратиться к шагу Q2.

Q9. [Сортировка методом простых вставок.] Для $j = 2, 3, \ldots, N$, если $K_{j-1} > K_j$, выполнять следующие операции: установить сначала $K \leftarrow K_j$, $R \leftarrow R_j$, $i \leftarrow j-1$, а затем — $R_{i+1} \leftarrow R_i$ и $i \leftarrow i-1$ один или более раз до тех пор, пока не выполнится условие $K_i \le K$. Далее установить $R_{i+1} \leftarrow R$. (Это, по существу, алгоритм 5.2.1S, модифицированный в соответствии с идеей упр. 5.2.1–10 и ответом к упр. 5.2.1–33. Шаг Q9 можно опустить, если $M = 1$. *Предупреждение.* Последняя стадия — сортировка методом простых вставок — может скрыть ошибки на шагах Q1–Q8, поэтому учтите, что программу, реализующую

этот алгоритм, следует тщательно проверить, не особо доверяя тому факту, что получен правильный результат!) ▮

Соответствующая MIX-программа довольно велика, но не сложна. На самом деле бóльшая часть команд относится к шагу Q7, на котором проводятся совершенно очевидные манипуляции переменными.

Программа Q (*Обменная сортировка с разделением*). Записи, которые предстоит рассортировать, находятся в ячейках от INPUT+1 до INPUT+N. Предполагается, что в ячейках INPUT и INPUT+N+1 содержатся значения, соответственно минимальное и максимальное для разрядной сетки компьютера MIX. Стек располагается в ячейках STACK+1, STACK+2, ...; точное число ячеек, которые необходимо отвести под стек, обсуждается в упр. 20. Значения регистров: rI2 $\equiv l$, rI3 $\equiv r$, rI4 $\equiv i$, rI5 $\equiv j$, rI6 $\equiv$ размер стека, rA $\equiv K \equiv R$. Полагаем, что $N > M$.

	A	EQU	2:3		Первый компонент элемента стека.
	B	EQU	4:5		Второй компонент элемента стека.
01	START	ENT6	0	1	*Q1. Начальная установка.* Очистить стек.
02		ENT2	1	1	$l \leftarrow 1$.
03		ENT3	N	1	$r \leftarrow N$.
04	2H	ENT5	1,3	A	*Q2. Начать новую итерацию.* $j \leftarrow r + 1$.
05		LDA	INPUT,2	A	$K \leftarrow K_l$.
06		ENT4	1,2	A	$i \leftarrow l + 1$.
07		JMP	0F	A	Перейти к шагу Q3, опустив "$i \leftarrow i + 1$".
08	6H	LDX	INPUT,4	B	*Q6. Взаимный обмен.*
09		ENT1	INPUT,4	B	
10		MOVE	INPUT,5	B	
11		STX	INPUT,5	B	$R_i \leftrightarrow R_j$.
12	3H	INC4	1	$C' - A$	*Q3. Сравнить $K_i : K$.* $i \leftarrow i + 1$.
13	0H	CMPA	INPUT,4	C'	
14		JG	3B	C'	Повторить, если $K > K_i$.
15	4H	DEC5	1	$C - C'$	*Q4. Сравнить $K : K_j$.* $j \leftarrow j - 1$.
16		CMPA	INPUT,5	$C - C'$	
17		JL	4B	$C - C'$	Повторить, если $K < K_j$.
18	5H	ENTX	0,5	$B + A$	*Q5. Проверить $i : j$.*
19		DECX	0,4	$B + A$	
20		JXP	6B	$B + A$	Перейти к шагу Q6, если $j > i$.
21		LDX	INPUT,5	A	
22		STX	INPUT,2	A	$R_l \leftarrow R_j$.
23		STA	INPUT,5	A	$R_j \leftarrow R$.
24	7H	ENT4	0,3	A	*Q7. Поместить в стек.*
25		DEC4	M,5	A	rI4 $\leftarrow r - j - M$.
26		ENT1	0,5	A	
27		DEC1	M,2	A	rI1 $\leftarrow j - l - M$.
28		ENTA	0,4	A	
29		DECA	0,1	A	
30		JANN	1F	A	Переход, если $r - j \geq j - l$.
31		J1NP	8F	A'	Перейти к шагу Q8, если $M \geq j - l > r - j$.
32		J4NP	3F	$S' + A''$	Переход, если $j - l > M \geq r - j$.
33		INC6	1	S'	(Сейчас $j - l > r - j > M$.)
34		ST2	STACK,6(A)	S'	

35		ENTA	-1,5	S'	
36		STA	STACK,6(B)	S'	$(l, j-1) \Rightarrow$ стек.
37	4H	ENT2	1,5	$S' + A'''$	$l \leftarrow j + 1$.
38		JMP	2B	$S' + A'''$	Перейти к шагу Q2.
39	1H	J4NP	8F	$A - A'$	Перейти к шагу Q8, если $M \geq r - j \geq$ $\geq j - l$.
40		J1NP	4B	$S - S' + A'''$	Переход, если $r - j > M \geq j - l$.
41		INC6	1	$S - S'$	(Сейчас $r - j \geq j - l > M$.)
42		ST3	STACK,6(B)	$S - S'$	
43		ENTA	1,5	$S - S'$	
44		STA	STACK,6(A)	$S - S'$	$(j+1, r) \Rightarrow$ стек.
45	3H	ENT3	-1,5	$S - S' + A''$	$r \leftarrow j - 1$.
46		JMP	2B	$S - S' + A''$	Перейти к шагу Q2.
47	8H	LD2	STACK,6(A)	$S + 1$	*Q8. Извлечение из стека.*
48		LD3	STACK,6(B)	$S + 1$	
49		DEC6	1	$S + 1$	$(l, r) \Leftarrow$ стек.
50		J6NN	2B	$S + 1$	Перейти к шагу Q2, если стек не пуст.
51	9H	ENT5	2-N	1	*Q9. Сортировка методом простых вставок.* $j \leftarrow 2$.
52	2H	LDA	INPUT+N,5	$N - 1$	$K \leftarrow K_j$, $R \leftarrow R_j$.
53		CMPA	INPUT+N-1,5	$N - 1$	(В этом цикле rI5 $\equiv j - N$.)
54		JGE	6F	$N - 1$	Переход, если $K \geq K_{j-1}$.
55	3H	ENT4	N-1,5	D	$i \leftarrow j - 1$.
56	4H	LDX	INPUT,4	E	
57		STX	INPUT+1,4	E	$R_{i+1} \leftarrow R_i$.
58		DEC4	1	E	$i \leftarrow i - 1$.
59		CMPA	INPUT,4	E	
60		JL	4B	E	Повторить, если $K < K_i$.
61	5H	STA	INPUT+1,4	D	$R_{i+1} \leftarrow R$.
62	6H	INC5	1	$N - 1$	
63		J5NP	2B	$N - 1$	$2 \leq j \leq N$. ▮

Анализ метода быстрой сортировки. Информацию о времени выполнения, приведенную вместе с программой Q, нетрудно вывести из закона сохранения Кирхгофа (см. раздел 1.3.3) и из того факта, что все помещенное в стек, в конце концов, оттуда извлекается. Закон Кирхгофа в применении к шагу Q2 показывает, что

$$A = 1 + (S' + A''') + (S - S' + A'') + S = 2S + 1 + A'' + A'''. \tag{15}$$

Следовательно, суммарное время реализации алгоритма составляет

$$24A + 11B + 4C + 3D + 8E + 7N + 9S \text{ машинных тактов,}$$

где

A — число итераций разбиения;
B — число взаимных обменов на шаге Q6;
C — число сравнений, выполненных во время разбиения;
D — число случаев, когда $K_{j-1} > K_j$ в ходе выполнения сортировки методом простых вставок (шаг Q9); (16)
E — количество инверсий, удаленных в процессе простых вставок;
S — количество случаев, когда происходит запись в стек.

Проанализировав эти шесть параметров, можно сделать обоснованный выбор значения параметра M, которым определяется "граница" между сортировкой методом простых вставок и методом разделения. Сам по себе анализ также весьма поучителен, поскольку алгоритм довольно сложен. В процессе анализа будут использоваться важные методики, которые читатели, освоившие их, смогут в дальнейшем применять самостоятельно. (Те же, кто не склонны к "разгребанию математических завалов", могут спокойно перевернуть страницы вплоть до формул (25).)

Как и при анализе большинства алгоритмов в этой главе, будем предполагать, что сортируемые ключи различны. В упр. 18 показано, что наличие равных ключей существенно не снижает эффективность алгоритма Q; в действительности присутствие равных ключей, по-видимому, даже способствует ее увеличению. Поскольку метод зависит только от относительного расположения ключей, можно предполагать также, что это просто числа $\{1, 2, \ldots, N\}$, расположенные в некотором порядке.

Справиться с проблемой попробуем, рассмотрев первую же итерацию разбиения, с которой мы впервые встречаемся на шаге Q7. Когда мы доберемся до этой операции, *подмассивы* $R_1 \ldots R_{j-1}$ *и* $R_{j+1} \ldots R_N$ *будут содержать записи в случайном порядке, если и в исходной последовательности они располагались в случайном порядке*, поскольку относительное положение записей в подмассивах никак не влияет на алгоритм разделения. Таким образом, комбинация последовательных разбиений может быть проанализирована посредством индукции по N. (Это важное замечание, поскольку альтернативные алгоритмы, которые не обладают этим свойством, оказываются, в конце концов, существенно более медленными; см. *Computing Surveys* **6** (1974), 287–289.)

Пусть s — значение первого ключа K_1, и предположим, что ровно t ключей $K_1, \ldots, K_s$ превосходят s. (Напомню, что сортируемые в наших примерах ключи — целые числа $\{1, 2, \ldots, N\}$.) Если $s = 1$, то нетрудно видеть, что произойдет во время первой итерации разделения. Шаг Q3 выполнится однократно, шаг Q4 — N раз, а шаг Q5 приведет нас к шагу Q7. Таким образом, "взнос" первой итерации в анализируемые параметры будет следующим: $A = 1$, $B = 0$, $C = N + 1$. Аналогичные, но несколько более сложные рассуждения для случая $s > 1$ (см. упр. 21) показывают, что взнос первой итерации в суммарное время выполнения в общем случае будет составлять

$$A = 1,\ B = t,\ C = N + 1 \qquad \text{для } 1 \le s \le N. \tag{17}$$

К этому необходимо добавить вклады последующих итераций, во время которых упорядочиваются подмассивы из $s - 1$ и $N - s$ элементов соответственно.

Если предположить, что в исходной последовательности записи располагались в случайном порядке, то можно выписать формулы, которыми определяются производящие функции для распределений вероятностей величин $A, B, \ldots, S$ (см. упр. 22). Но для простоты рассмотрим здесь только *средние* значения этих величин A_N, $B_N, \ldots, S_N$ как функции от N. Рассмотрим, например, среднее число сравнений C_N, выполненных в процессе разделения. Если $N \le M$, то $C_N = 0$. В противном случае, поскольку любое данное значение s встречается с вероятностью $1/N$, получим

$$C_N = \frac{1}{N}\sum_{s=1}^{N}(N+1+C_{s-1}+C_{N-s})$$

$$= N+1+\frac{2}{N}\sum_{0\le k<N} C_k \qquad \text{для } N > M. \tag{18}$$

Аналогичные формулы имеют место и для остальных величин A_N, B_N, D_N, E_N, S_N (см. упр. 23).

Существует простой способ решения рекуррентных соотношений вида

$$x_n = f_n + \frac{2}{n}\sum_{0\le k<n} x_k \qquad \text{для } n \ge m. \tag{19}$$

На первом шаге нужно освободиться от знака суммирования: поскольку

$$(n+1)x_{n+1} = (n+1)f_{n+1} + 2\sum_{0\le k\le n} x_k,$$

$$nx_n = nf_n + 2\sum_{0\le k<n} x_k,$$

можно вычесть второе равенство из первого, получив

$$(n+1)x_{n+1} - nx_n = g_n + 2x_n, \qquad \text{где } g_n = (n+1)f_{n+1} - nf_n.$$

Теперь рекуррентная зависимость принимает гораздо более простой вид:

$$(n+1)x_{n+1} = (n+2)x_n + g_n \qquad \text{для } n \ge m. \tag{20}$$

Любое рекуррентное соотношение общего вида

$$a_n x_{n+1} = b_n x_n + g_n \tag{21}$$

можно свести к простой сумме, если умножить обе части на "суммирующий множитель" $a_0\, a_1 \ldots a_{n-1}/b_0\, b_1 \ldots b_n$. Получаем

$$y_{n+1} = y_n + c_n, \quad \text{где} \quad y_n = \frac{a_0 \ldots a_{n-1}}{b_0 \ldots b_{n-1}}\, x_n, \quad c_n = \frac{a_0 \ldots a_{n-1}}{b_0\, b_1 \ldots b_n}\, g_n. \tag{22}$$

Для (20) суммирующий множитель равен просто $n!/(n+2)! = 1/(n+1)(n+2)$. Таким образом, находим, что простое соотношение

$$\frac{x_{n+1}}{n+2} = \frac{x_n}{n+1} + \frac{(n+1)f_{n+1} - nf_n}{(n+1)(n+2)} \qquad \text{для } n \ge m \tag{23}$$

является следствием соотношения (19).

Если, например, положить $f_n = 1/n$, то получится неожиданный результат: $x_n/(n+1) = x_m/(m+1)$ при любом $n \ge m$. Если положить $f_n = n+1$, получится

$$\begin{aligned} x_n/(n+1) &= 2/(n+1) + 2/n + \cdots + 2/(m+2) + x_m/(m+1) \\ &= 2\,(H_{n+1} - H_{m+1}) + x_m/(m+1) \end{aligned}$$

при любом $n \geq m$. Таким образом получим решение для (18), подставив $m = M + 1$ и $x_n = 0$ при $n \leq M$; искомая формула имеет вид

$$\begin{aligned} C_N &= (N+1)\,(2H_{N+1} - 2H_{M+2} + 1) \\ &\approx 2\,(N+1)\,\ln\left(\frac{N+1}{M+2}\right) \qquad \text{для } N > M. \end{aligned} \tag{24}$$

В упр. 6.2.2–8 будет доказано, что при $M = 1$ стандартное отклонение величины C_N асимптотически приближается к $\sqrt{(21 - 2\pi^2)/3}\,N$. Это довольно мало по сравнению с (24).

Остальные величины можно найти аналогичным способом (см. упр. 23); при $N > M$ имеем

$$\begin{aligned} A_N &= 2\,(N+1)/(M+2) - 1, \\ B_N &= \frac{1}{6}\,(N+1)\left(2H_{N+1} - 2H_{M+2} + 1 - \frac{6}{M+2}\right) + \frac{1}{2}, \\ D_N &= (N+1)\big(1 - 2H_{M+1}/(M+2)\big), \\ E_N &= \tfrac{1}{6}\,(N+1)M(M-1)/(M+2); \\ S_N &= (N+1)/(2M+3) - 1 \qquad \text{для } N > 2M+1. \end{aligned} \tag{25}$$

Из приведенных выше соображений следует, что можно точно проанализировать среднее время выполнения весьма сложной программы, используя методы, которые мы ранее применяли в более простых случаях.

Чтобы определить наилучшее значение M для конкретного компьютера, можно воспользоваться формулами (24) и (25). При $N > 2M + 1$ выполнение программы Q на компьютере MIX требует в среднем $(35/3)(N+1)H_{N+1} + \frac{1}{6}(N+1)f(M) - 34.5$ машинных циклов, где

$$f(M) = 8M - 70H_{M+2} + 71 - 36\,\frac{H_{M+1}}{M+2} + \frac{270}{M+2} + \frac{54}{2M+3}\,. \tag{26}$$

Желательно выбрать такое значение M, при котором функция $f(M)$ достигает минимума. Несложные рассуждения приводят нас к такому результату: $M = 9$. При этом для больших N среднее время выполнения программы Q равно приблизительно $11.667(N1+)\ln N - 1.74N - 18.74$ машинных циклов.

Таким образом, программа Q работает в среднем довольно быстро; следует, кроме того, учесть, что она требует очень мало памяти. Высокая скорость определяется, в первую очередь, тем, что внутренний цикл — шаги Q3 и Q4 — очень короток (см. строки 12–14 и 15–17 текста программы). Количество операций обмена записей в памяти на шаге Q6 составляет всего около 1/6 количества сравнений на шагах Q3 и Q4; к тому же мы очень много экономим вследствие того, что отпала необходимость в сравнении i с j во внутреннем цикле.

Но каков *наихудший* случай для алгоритма Q? Существуют ли какие-нибудь исходные файлы, обрабатывать которые с помощью этого алгоритма не эффективно? Ответ несколько обескураживает: если исходный файл уже упорядочен, а именно — $K_1 < K_2 < \cdots < K_N$, то каждая операция "разделения" становится почти бесполезной, так как она уменьшает размер подмассива всего лишь на один элемент! Значит,

в этой ситуации (которая должна быть самой простой для сортировки) быстрая сортировка превращается в отнюдь не быструю. Время ее работы становится пропорциональным N^2, а не $N \lg N$ (см. упр. 25). В отличие от других алгоритмов сортировки, которые нам встречались, алгоритм Q *предпочитает* неупорядоченные файлы!

В упомянутой статье Хоара предложены два способа устранения этого недостатка. Они базируются на выборе лучшего значения проверяемого ключа K, который управляет разделением. Одна из его рекомендаций состоит в том, чтобы в последней части шага Q2 выбрать *случайное* целое число q в диапазоне между l и r. На этом шаге можно заменить команды выполнения операций $K \leftarrow K_l$ операциями

$$K \leftarrow K_q, \qquad R \leftarrow R_q, \qquad R_q \leftarrow R_l, \qquad R_l \leftarrow R. \tag{27}$$

(Последняя операция присваивания необходима, поскольку в противном случае на шаге Q4 произойдет прекращение цикла с сохранением $j = l - 1$, если K есть наименьший ключ в разделяемом подмассиве.) Согласно формулам (25) такие случайные целые числа придется вычислять в среднем лишь $2\,(N+1)/(M+2) - 1$ раз, так что дополнительные расходы несущественны, а случайный выбор — хорошая защита от возможности оказаться в наихудшей ситуации. Даже "слабая" случайность в выборе q может служить достаточной защитой. В упр. 58 доказано, что при действительно случайном значении q появление более чем $20N \ln N$ сравнений возможно с вероятностью, меньшей 10^{-8}.

Второе предложение Хоара состоит в просмотре небольшого фрагмента сортируемой последовательности и нахождении медианы для этой совокупности данных. Такому подходу последовал Р. К. Синглтон [см. R. C. Singleton, *CACM* **12** (1969), 185–187], который предложил в качестве K_q брать медиану трех значений:

$$K_l, \qquad K_{\lfloor (l+r)/2 \rfloor}, \qquad K_r. \tag{28}$$

Процедура Синглтона сокращает число сравнений с $2N \ln N$ примерно до $\frac{12}{7} N \ln N$ (см. упр. 29). Можно показать, что в этом случае B_N асимптотически приближается к $C_N/5$, а не к $C_N/6$, так что метод медианы несколько увеличивает время, затрачиваемое на пересылку данных. В результате общее время работы сокращается примерно на 8%. (Подробный анализ приводится в упр. 56.) Время работы в наихудшем случае все еще составляет порядка N^2, однако с таким случаем вряд ли когда-либо придется сталкиваться на практике. У. Д. Фрэйзер и А. Ч. Мак-Келлар [см. W. D. Frazer and A. C. McKellar, *JACM* **17** (1970), 496–507] предложили рассматривать совокупность гораздо большего объема из $2^k - 1$ записей, где k выбирается так, чтобы $2^k \approx N/\ln N$. Эту совокупность можно рассортировать обычным методом быстрой сортировки, после чего элементы расположить среди остальных записей за k просмотров всего массива (в результате массив записей будет разделен на 2^k подмассивов, ограниченных элементами выборки). На заключительном этапе сортируются полученные подмассивы. Если N находится в реальном диапазоне значений, то среднее число сравнений, выполняемых такой процедурой "сортировки выборки", примерно такое же, как и для метода медианы Синглтона, но при $N \to \infty$ оно асимптотически приближается к $N \lg N$.

Метод, который даст абсолютную гарантию того, что в худшем случае время сортировки будет составлять порядка $O(N \log N)$ и одновременно будет обеспечена высокая скорость в среднем, можно получить, комбинируя метод быстрой сорти-

ровки с некоторыми другими схемами. Например, Д. Р. Муссер [см. D. R. Musser *Software Practice & Exper.* **27** (1997), 983–993] предложил ввести в каждый элемент стека быстрой сортировки еще и компонент "глубина разделения". Если найден подмассив, который разделен более чем, скажем, $2\lg N$ раз, предлагается прекратить выполнение алгоритма Q и переключиться на алгоритм 5.2.3H. В таком случае сохраняется то же время выполнения внутреннего цикла, а следовательно, и среднее время выполнения сортировки.

Роберт Седгевик проанализировал множество оптимизированных вариантов быстрой сортировки в *Acta Informatica* **7** (1977), 327–356, и в *CACM* **21** (1978), 847–857, **22** (1979), 368. Версии быстрой сортировки, включенные в библиотеку программ UNIX®, которая создавалась в течение 15 лет, описана в работе J. L. Bentley and M. D. McIlroy, *Software Practice & Exper.* **23** (1993), 1249–1265.

Обменная поразрядная сортировка. Рассмотрим метод, совершенно отличный от всех схем сортировки, которые описывались до сих пор. В нем используется *двоичное представление* ключей, и потому он предназначен только для двоичных компьютеров. Вместо того чтобы сравнивать между собой два ключа, в этом методе проверяется, равны ли 0 или 1 отдельные разряды ключа. В других отношениях он обладает характеристиками обменной сортировки и на самом деле очень напоминает быструю сортировку. Так как метод использует разряды ключа, представленного в двоичной системе счисления, назовем его обменной поразрядной сортировкой. В общих чертах этот алгоритм можно описать следующим образом.

i) Последовательность сортируется по *старшему значащему двоичному разряду* так, чтобы все ключи, начинающиеся с 0, оказались перед всеми ключами, начинающимися с 1. Для этого необходимо найти крайний слева ключ K_i, начинающийся с 1, и крайний справа ключ K_j, начинающийся с 0, после чего R_i и R_j поменяются местами, и процесс будет повторяться, пока не получится $i > j$.

ii) Пусть F_0 — множество элементов, начинающихся с 0, а F_1 — множество всех остальных элементов. Будем применять к F_0 поразрядную сортировку (начав теперь со *второго* бита слева, а не со старшего) до тех пор, пока множество F_0 полностью не рассортируется. Затем проделаем то же самое с F_1.

Например, в табл. 3 показано, как действует обменная поразрядная сортировка на наши 16 случайных чисел, записанных теперь в восьмеричной системе счисления. В строке итерации 1 показан исходный массив; после обменов по первому биту приходим ко второй итерации. На второй итерации сортируется первая группа по второму биту, на третьей — по третьему биту. (Читатель должен мысленно преобразовать восьмеричные числа в 10-разрядные двоичные. Например, *0232* в двоичном коде будет равно $(0\,010\,011\,010)_2$.) Достигнув после сортировки по четвертому биту пятой итерации, обнаруживаем, что в каждой из оставшихся групп содержится всего по одному элементу, так что эту часть массива можно больше не рассматривать. Запись "^{4}[*0232 0252*]" означает, что подмассив *0232 0252* еще предстоит сортировать по четвертому биту слева. В этом конкретном случае сортировка по четвертому биту не дает ничего нового; чтобы разделить элементы, необходимо добраться до пятого бита.

Весь процесс сортировки, показанный в табл. 3, выполняется за 22 итерации; это несколько больше соответствующего значения при быстрой сортировке (см. табл. 2).

Количество операций анализа отдельных разрядов — 82 — также велико, но мы увидим, что число такого типа операций при больших N в действительности меньше, чем число сравнений при быстрой сортировке, в предположении, что значения ключей имеют равномерное распределение. Общее количество обменов записей в табл. 3 равно 17, т. е. оно весьма умеренно. Заметим, что, хотя сортируются 10-битовые числа, в данном примере при проверке битов никогда не приходится идти дальше седьмого бита.

Как и при быстрой сортировке, для хранения "информации о границах" подмассивов, ожидающих сортировки, можно воспользоваться стеком. Вместо того чтобы сортировать, в первую очередь, наименьший из подмассивов, удобно просто продвигаться слева направо, и размер стека в этом случае никогда не превзойдет числа разрядов в двоичном представлении сортируемых ключей. В алгоритме, приведенном ниже, элемент стека (r, b) указывает на то, что подмассив с правой границей r ожидает сортировки по b-му разряду; левую границу можно не запоминать в стеке: она всегда задана неявно, поскольку в этой процедуре массив всегда обрабатывается слева направо.

Алгоритм R (*Обменная поразрядная сортировка*). Записи $R_1, \ldots, R_N$ перекомпоновываются в пределах той же области памяти. По завершении сортировки их ключи будут упорядочены: $K_1 \leq \cdots \leq K_N$. Предполагается, что все ключи — m-разрядные двоичные числа $(a_1\, a_2 \ldots a_m)_2$; i-й по старшинству разряд a_i называется "разряд i" ключа. Необходим вспомогательный стек, вмещающий до $m - 1$ элементов. Этот алгоритм, по существу, следует процедуре обменной поразрядной сортировки с разделениями, описанной выше. Возможны некоторые усовершенствования с целью повышения эффективности (они описаны ниже, в тексте раздела и в упражнениях).

R1. [Начальная установка.] Очистить стек и установить $l \leftarrow 1$, $r \leftarrow N$, $b \leftarrow 1$.

R2. [Начать новую итерацию.] (Теперь желательно рассортировать подмассив $R_l \cdots R_r$ по разряду b в соответствии с алгоритмом $l \leq r$.) Если $l = r$, перейти к шагу R10 (так как массив, состоящий из одного слова, уже рассортирован). В противном случае установить $i \leftarrow l$, $j \leftarrow r$.

R3. [Проверить наличие 1 в K_i.] Проверить разряд b ключа K_i. Если он равен 1, то перейти к шагу R6.

R4. [Увеличить i.] Увеличить i на 1. Если $i \leq j$, возвратиться к шагу R3; в противном случае перейти к шагу R8.

R5. [Проверить наличие 0 в K_{j+1}.] Проверить разряд b ключа K_{j+1}. Если он равен 0, то перейти к шагу R7.

R6. [Уменьшить j.] Уменьшить j на 1. Если $i \leq j$, перейти к шагу R5; в противном случае перейти к шагу R8.

R7. [Поменять местами R_i, R_{j+1}.] Поменять местами записи $R_i \leftrightarrow R_{j+1}$ и перейти к шагу R4.

R8. [Проверить особые случаи.] (К этому моменту итерация разделения завершена, $i = j + 1$, разряд b ключей $K_l, \ldots, K_j$ равен 0, а разряд b ключей $K_i, \ldots, K_r$ равен 1.) Увеличить b на 1. Если $b > m$, где m — общее число разрядов в ключах, перейти к шагу R10. (Это означает, что подмассив $R_l \ldots R_r$ рассортирован. Если в массиве не может быть равных ключей, то такую проверку

Таблица 3

ОБМЕННАЯ ПОРАЗРЯДНАЯ СОРТИРОВКА

Итерация																	l	r	b	Стек
1	^{1}⟦0767	0127	1000	0075	1614	0252	1601	0423	1215	0652	0232	0775	1144	1245	1375	1277⟧	1	16	1	—
2	^{2}⟦0767	0127	0775	0075	0232	0252	0652	0423⟧	^{2}[1215	1601	1614	1000	1144	1245	1375	1277]	1	8	2	(16,2)
3	^{3}⟦0252	0127	0232	0075⟧	^{3}[0775	0767	0652	0423]	^{2}[1215	1601	1614	1000	1144	1245	1375	1277]	1	4	3	(8,3)(16,2)
4	^{4}⟦0075	0127⟧	^{4}[0232	0252]	^{3}[0775	0767	0652	0423]	^{2}[1215	1601	1614	1000	1144	1245	1375	1277]	1	2	4	(4,4)(8,3)(16,2)
5	0075	0127	^{4}⟦0232	0252⟧	^{3}[0775	0767	0652	0423]	^{2}[1215	1601	1614	1000	1144	1245	1375	1277]	3	4	4	(8,3)(16,2)
6	0075	0127	^{5}⟦0232	0252⟧	^{3}[0775	0767	0652	0423]	^{2}[1215	1601	1614	1000	1144	1245	1375	1277]	3	4	5	(8,3)(16,2)
7	0075	0127	0232	0252	^{3}⟦0775	0767	0652	0423⟧	^{2}[1215	1601	1614	1000	1144	1245	1375	1277]	5	8	3	(16,2)
8	0075	0127	0232	0252	0423	^{4}⟦0767	0652	0775⟧	^{2}[1215	1601	1614	1000	1144	1245	1375	1277]	6	8	4	(16,2)
9	0075	0127	0232	0252	0423	0652	^{5}⟦0767	0775⟧	^{2}[1215	1601	1614	1000	1144	1245	1375	1277]	7	8	5	(16,2)
10	0075	0127	0232	0252	0423	0652	^{6}⟦0767	0775⟧	^{2}[1215	1601	1614	1000	1144	1245	1375	1277]	7	8	6	(16,2)
11	0075	0127	0232	0252	0423	0652	^{7}⟦0767	0775⟧	^{2}[1215	1601	1614	1000	1144	1245	1375	1277]	7	8	7	(16,2)
12	0075	0127	0232	0252	0423	0652	0767	0775	^{2}⟦1215	1601	1614	1000	1144	1245	1375	1277⟧	9	16	2	—
13	0075	0127	0232	0252	0423	0652	0767	0775	^{3}⟦1215	1277	1375	1000	1144	1245⟧	^{3}[1614	1601]	9	14	3	(16,3)
14	0075	0127	0232	0252	0423	0652	0767	0775	^{4}⟦1144	1000⟧	^{4}[1375	1277	1215	1245]	^{3}[1614	1601]	9	10	4	(14,4)(16,3)
15	0075	0127	0232	0252	0423	0652	0767	0775	1000	1144	^{4}⟦1375	1277	1215	1245⟧	^{3}[1614	1601]	11	14	4	(16,3)
16	0075	0127	0232	0252	0423	0652	0767	0775	1000	1144	^{5}⟦1245	1277	1215⟧	^{5}[1375]	^{3}[1614	1601]	11	13	5	(14,5)(16,3)
17	0075	0127	0232	0252	0423	0652	0767	0775	1000	1144	1215	^{6}⟦1277	1245⟧	^{5}[1375]	^{3}[1614	1601]	12	13	6	(14,5)(16,3)
18	0075	0127	0232	0252	0423	0652	0767	0775	1000	1144	1215	1245	1277	1375	^{3}⟦1614	1601⟧	15	16	3	—
19	0075	0127	0232	0252	0423	0652	0767	0775	1000	1144	1215	1245	1277	1375	^{4}⟦1614	1601⟧	15	16	4	—
20	0075	0127	0232	0252	0423	0652	0767	0775	1000	1144	1215	1245	1277	1375	^{5}⟦1614	1601⟧	15	16	5	—
21	0075	0127	0232	0252	0423	0652	0767	0775	1000	1144	1215	1245	1277	1375	^{6}⟦1614	1601⟧	15	16	6	—
22	0075	0127	0232	0252	0423	0652	0767	0775	1000	1144	1215	1245	1277	1375	^{7}⟦1614	1601⟧	15	16	7	—
23	0075	0127	0232	0252	0423	0652	0767	0775	1000	1144	1215	1245	1277	1375	1601	1614	17	–	–	—

При поразрядной обменной сортировке для получения конечного результата нужно всего один раз проанализировать каждый разряд.

можно не делать.) В противном случае, если $j < l$ или $j = r$, возвратиться к шагу R2 (все просмотренные разряды оказались равными соответственно 1 или 0). Если же $j = l$, то увеличить l на 1 и перейти к шагу R2 (встретился только один разряд, равный 0).

R9. [Поместить в стек.] Поместить в стек элемент (r, b), а затем установить $r \leftarrow j$ и перейти к шагу R2.

R10. [Извлечение из стека.] Если стек пуст, значит, сортировка завершена. В противном случае установить $l \leftarrow r + 1$, извлечь из стека элемент (r', b'), установить $r \leftarrow r'$, $b \leftarrow b'$ и возвратиться к шагу R2. ▮

Программа R (*Обменная поразрядная сортировка*). В следующей программе для машины MIX используются, по существу, те же соглашения, что и в программе Q. Значения регистров таковы: rI1 $\equiv l - r$, rI2 $\equiv r$, rI3 $\equiv i$, rI4 $\equiv j$, rI5 $\equiv m - b$, rI6 $\equiv$ размер стека, если не учитывать, что в некоторых командах (отмеченных ниже) удобно оставить rI3 $= i - j$ или rI4 $= j - i$. Двоичная природа поразрядной сортировки объясняет использование в этой программе команд SRB (поразрядный сдвиг содержимого AX вправо), JAE (переход, если содержимое A четно) и JAO (переход, если содержимое A нечетно), определенных в разделе 4.5.2. Предполагается, что $N \geq 2$.

01	START	ENT6	0	1	*R1. Начальная установка.* Очистить стек.	
02		ENT1	1-N	1	$l \leftarrow 1$.	
03		ENT2	N	1	$r \leftarrow N$.	
04		ENT5	M-1	1	$b \leftarrow 1$.	
05		JMP	1F	1	Перейти к шагу R2 (опустить проверку $l = r$).	
06	9H	INC6	1	S	*R9. Поместить в стек.*	[rI4 $= l - j$]
07		ST2	STACK,6(A)	S		
08		ST5	STACK,6(B)	S	$(r, b) \Rightarrow$ стек.	
09		ENN1	0,4	S	rI1 $\leftarrow l - j$.	
10		ENT2	-1,3	S	$r \leftarrow j$.	
11	1H	ENT3	0,1	A	*R2. Начать новую итерацию.*	[rI3 $= i - j$]
12		ENT4	0,2	A	$i \leftarrow l$, $j \leftarrow r$.	[rI3 $= i - j$]
13	3H	INC3	0,4	C'	*R3. Проверить наличие 1 в K_i.*	
14		LDA	INPUT,3	C'		
15		SRB	0,5	C'	Младший разряд rA $\leftarrow$ разряд b ключа K_i.	
16		JAE	4F	C'	Перейти к шагу R4, если он равен 0.	
17	6H	DEC4	1,3	$C'' + X$	*R6. Уменьшить j.* $j \leftarrow j - 1$.	[rI4 $= j - i$]
18		J4N	8F	$C'' + X$	Перейти к шагу R8, если $j < i$.	[rI4 $= j - i$]
19	5H	INC4	0,3	C''	*R5. Проверить наличие 0 в K_{j+1}.*	
20		LDA	INPUT+1,4	C''		
21		SRB	0,5	C''	Младший разряд rA $\leftarrow$ разряд b ключа K_{j+1}.	
22		JAO	6B	C''	Перейти к шагу R6, если он равен 1.	
23	7H	LDA	INPUT+1,4	B	*R7. Поменять местами R_i, R_{j+1}.*	
24		LDX	INPUT,3	B		
25		STX	INPUT+1,4	B		
26		STA	INPUT,3	B		
27	4H	DEC3	-1,4	$C' - X$	*R4. Увеличить i.* $i \leftarrow i + 1$.	[rI3 $= i - j$]
28		J3NP	3B	$C' - X$	Перейти к шагу R3, если $i \leq j$.	[rI3 $= i - j$]
29		INC3	0,4	$A - X$	rI3 $\leftarrow i$.	

30	8H	J5Z	0F	A	*R8. Проверить особые случаи.*[rI4 неизвестно]
31		DEC5	1	$A-G$	Перейти к шагу R10, если $b=m$, иначе — $b \leftarrow b-1$.
32		ENT4	-1,3	$A-G$	rI4 $\leftarrow j$.
33		DEC4	0,2	$A-G$	rI4 $\leftarrow j-r$.
34		J4Z	1B	$A-G$	Перейти к шагу R2, если $j=r$.
35		DEC4	0,1	$A-G-R$	rI4 $\leftarrow j-l$.
36		J4N	1B	$A-G-R$	Перейти к шагу R2, если $j<l$.
37		J4NZ	9B	$A-G-L-R$	Перейти к шагу R9, если $j \neq l$.
38		INC1	1	K	$l \leftarrow l+1$.
39	2H	J1NZ	1B	$K+S$	Переход, если $l \neq r$.
40	0H	ENT1	1,2	$S+1$	*R10. Извлечение из стека.*
41		LD2	STACK,6(A)	$S+1$	
42		DEC1	0,2	$S+1$	
43		LD5	STACK,6(B)	$S+1$	Стек $\Rightarrow (r,b)$.
44		DEC6	1	$S+1$	
45		J6NN	2B	$S+1$	Перейти к шагу R2, если стек не пуст. ▌

Время работы программы обменной поразрядной сортировки зависит от следующих показателей:

$$
\begin{aligned}
&A \text{ — число итераций, в которых } l<r;\\
&B \text{ — число операций обмена записей;}\\
&C=C'+C'' \text{ — число проверок разрядов;}\\
&G \text{ — число случаев, когда } b>m \text{ на шаге R8;}\\
&K \text{ — число случаев, когда } b \le m,\ j=l \text{ на шаге R8;}\\
&L \text{ — число случаев, когда } b \le m,\ j<l \text{ на шаге R8;}\\
&R \text{ — число случаев, когда } b \le m,\ j=r \text{ на шаге R8;}\\
&S \text{ — число случаев, когда что-либо помещается в стек;}\\
&X \text{ — число случаев, когда } j<i \text{ на шаге R6.}
\end{aligned}
\tag{29}
$$

По закону Кирхгофа $S=A-G-K-L-R$, так что общее время выполнения равно $27A+8B+8C-23G-14K-17L-19R-X+13$ машинных циклов. За счет усложнения программы можно несколько ускорить циклы проверки разрядов (см. упр. 34). Обменную поразрядную сортировку можно также ускорить, если при достаточно малых значениях разности $r-l$ применять простые вставки, как это сделано в алгоритме Q, но мы не будем задерживаться на таких усовершенствованиях. Проанализируем время выполнения обменной поразрядной сортировки в двух случаях, которым соответствуют разные наборы условий.

i) Пусть $N=2^m$ и ключи, которые следует рассортировать, — просто числа 0, 1, 2, ..., 2^m-1, расположенные в случайном порядке.

ii) Пусть $m=\infty$ (неограниченная точность) и ключи, которые необходимо рассортировать, — независимые, равномерно распределенные в промежутке $[0\,..\,1)$ действительные числа.

Анализ случая (i) относительно прост, поэтому он оставлен читателю в качестве упражнения (см. упр. 35). Случай (ii) более сложен, но его мы также оставили для упражнений. В следующей таблице приведены грубые оценки результатов анализа в том и другом случаях.

Величина	Случай (i)	Случай (ii)
A	N	αN
B	$\frac{1}{4}N\lg N$	$\frac{1}{4}N\lg N$
C	$N\lg N$	$N\lg N$
G	$\frac{1}{2}N$	0
K	0	$\frac{1}{2}N$
L	0	$\frac{1}{2}(\alpha-1)N$
R	0	$\frac{1}{2}(\alpha-1)N$
S	$\frac{1}{2}N$	$\frac{1}{2}N$
X	$\frac{1}{2}N$	$\frac{1}{4}(\alpha+1)N$ (30)

Здесь $\alpha = 1/\ln 2 \approx 1.4427$. Заметим, что среднее число обменов, проверок разрядов и обращений к стеку, по существу, одинаково в обоих случаях, несмотря даже на то, что в случае (ii) число итераций на 44% больше. На сортировку N элементов в случае (ii) наша MIX-программа затратит в среднем приблизительно $14.4\,N\ln N$ машинных циклов. Это число можно сократить примерно до $11.5\,N\ln N$, если воспользоваться предложением из упр. 34. Соответствующая величина для программы Q равна $11.7\,N\ln N$, но и ее можно уменьшить до $10.6\,N\ln N$, если воспользоваться предложением Синглтона и выбрать медиану из трех ключей.

В случае равномерного распределения данных обменная поразрядная сортировка отнимает в среднем примерно столько же времени, сколько и быстрая сортировка; на некоторых компьютерах она выполняется немного быстрее, чем быстрая сортировка. В упр. 53 показано, в какой мере замедляется этот процесс при неравномерном распределении. Важно отметить, что весь наш анализ основан на предположении о том, что все ключи различны. *Обменная поразрядная сортировка в таком виде, как описано выше, не очень эффективна, если имеются одинаковые ключи*, поскольку она проходит через несколько итераций, каждая из которых требует определенного времени для разделения множества одинаковых ключей до тех пор, пока b не станет $> m$. Один приемлемый способ исправления этого недостатка предложен в ответе к упр. 40.

Как обменная поразрядная сортировка, так и быстрая сортировка основаны на идее разбиения. Записи меняются местами до тех пор, пока массив не будет разбит на две части: левый подмассив, в котором все ключи $\le K$ при определенном K, и правый подмассив, в котором все ключи $\ge K$. При быстрой сортировке в качестве K выбирается реальный ключ из массива, в то время как при обменной поразрядной сортировке, по существу, выбирается некоторый искусственный ключ на основе двоичных представлений. Что касается исторической стороны дела, то обменную поразрядную сортировку открыли П. Хильдебрандт, Г. Исбитц, Х. Райзинг и Ж. Шварц [см. P. Hildebrandt, H. Isbitz, H. Rising, J. Schwartz, *JACM* **6** (1959), 156–163] примерно за год до изобретения быстрой сортировки. Существуют и другие схемы разделения; например, Джон Мак-Карти (John McCarthy) предложил выбирать $K \approx \frac{1}{2}(u+v)$, если известно, что все ключи находятся в диапазоне между u и v.

Еще одну стратегию разделения предложил М. Х. ван Эмден (M. H. van Emden) [*CACM* **13** (1970), 563–567]: вместо того чтобы выбирать K заранее, мы "узнаем",

каким может быть хорошее значение K, отслеживая в процессе разделения изменение величин $K' = \max(K_l, \ldots, K_i)$ и $K'' = \min(K_j, \ldots, K_r)$. Можно увеличивать i до тех пор, пока не встретится ключ, больший K', а затем начать уменьшать j, пока не встретится ключ, меньший K'', после чего поменять их местами и/или уточнить значения K' и K''. Эмпирические тесты этой "интервальной сортировки с обменом" показывают, что она осуществляется несколько медленнее, чем быстрая, а теоретический анализ выполнить очень сложно. В значительной мере затруднение во время анализа вызывает тот факт, что записи в подмассивах после разделения уже нельзя считать размещенными в случайном порядке. Это единственный метод, обсуждаемый в данной книге, для поведения которого еще не найдено адекватного теоретического объяснения.

Обобщение обменной поразрядной сортировки для системы счисления с основанием, большим 2, обсуждается в разделе 5.2.5.

***Асимптотические методы.** Анализ алгоритмов обменной сортировки приводит к некоторым особенно поучительным математическим задачам, которые позволяют больше узнать о способах определения асимптотического поведения функции. Например, при анализе метода пузырька [формула (9)] мы столкнулись с функцией

$$W_n = \frac{1}{n!} \sum_{0 \le r < s \le n} s!\, r^{n-s}. \tag{31}$$

Какой вид имеет асимптотическое выражение для этой функции?

Можно действовать так же, как при исследовании числа инволюций [формула 5.1.4–(41)]. Поэтому, прежде чем двигаться дальше, полезно еще раз просмотреть материал, изложенный в конце раздела 5.1.4. Исследование формулы (31) показывает, что вклад при $s = n$ больше, чем вклад при $s = n - 1$ и т. д. Это наводит на мысль о замене s на $n - s$. Мы скоро увидим, что на самом деле удобнее всего применить подстановки $t = n - s + 1$, $m = n + 1$, в результате которых формула (31) примет вид

$$\frac{1}{m} W_{m-1} = \frac{1}{m!} \sum_{1 \le t < m} (m-t)! \sum_{0 \le r < m-t} r^{t-1}. \tag{32}$$

Для внутренней суммы существует хорошо известный асимптотический ряд, полученный из формулы суммирования Эйлера:

$$\begin{aligned} \sum_{0 \le r < N} r^{t-1} &= \frac{N^t}{t} - \frac{1}{2}(N^{t-1} - \delta_{t1}) + \frac{B_2}{2!}(t-1)(N^{t-2} - \delta_{t2}) + \cdots \\ &= \frac{1}{t} \sum_{j=0}^{k} \binom{t}{j} B_j (N^{t-j} - \delta_{tj}) + O(N^{t-k}) \end{aligned} \tag{33}$$

(см. упр. 1.2.11.2–4). Следовательно, наша задача сводится к анализу сумм вида

$$\frac{1}{m!} \sum_{1 \le t < m} (m-t)!\,(m-t)^t t^k, \qquad k \ge -1. \tag{34}$$

Как и в разделе 5.1.4, можно показать, что при значениях t, больших $m^{1/2+\epsilon}$, члены суммы пренебрежимо малы: $O(\exp(-n^\delta))$. Значит, можно положить $t = O(m^{1/2+\epsilon})$ и заменить факториалы по формуле Стирлинга:

$$\frac{(m-t)!\,(m-t)^t}{m!}$$
$$= \sqrt{1-\frac{t}{m}}\,\exp\left(\frac{t}{12m^2} - \left(\frac{t^2}{2m} + \frac{t^3}{3m^2} + \frac{t^4}{4m^3} + \frac{t^5}{5m^4}\right) + O(m^{-2+6\epsilon})\right).$$

Таким образом, нас интересует асимптотическое выражение для суммы

$$r_k(m) = \sum_{1\le t<m} e^{-t^2/2m} t^k, \qquad k \ge -1. \tag{35}$$

(Суммирование здесь также можно было бы распространить на весь диапазон $1 \le t < \infty$, не изменив при этом значения асимптотического выражения суммы, поскольку, как указано выше, входящие в сумму значения при $t > m^{1/2+\epsilon}$ пренебрежимо малы.)

Пусть $g_k(x) = x^k e^{-x^2}$ и $f_k(x) = g_k\bigl(x/\sqrt{2m}\bigr)$. По формуле суммирования Эйлера при $k \ge 0$ получим

$$\sum_{0\le t<m} f_k(t) = \int_0^m f_k(x)\,dx + \sum_{j=1}^{p} \frac{B_j}{j!}\left(f_k^{(j-1)}(m) - f_k^{(j-1)}(0)\right) + R_p,$$
$$R_p = \frac{(-1)^{p+1}}{p!}\int_0^m B_p(\{x\}) f_k^{(p)}(x)\,dx$$
$$= \left(\frac{1}{\sqrt{2m}}\right)^p O\left(\int_0^\infty \bigl|g_k^{(p)}(y)\bigr|\,dy\right) = O(m^{-p/2}). \tag{36}$$

Следовательно, при помощи, по существу, тех же приемов, которые применялись и раньше, можно получить асимптотический ряд для $r_k(m)$, если только $k \ge 0$. Но при $k = -1$ этот метод не годится, так как значение $f_{-1}(0)$ не определено; нельзя также просто просуммировать от 1 до m, так как остаточные члены не дают убывающих степеней m, если нижний предел равен 1. (Именно в этом состоит суть дела, и, прежде чем двигаться дальше, читатель должен убедиться в том, что он хорошо понял задачу.)

Чтобы разрешить эту дилемму, можно положить по определению $g_{-1}(x) = (e^{-x^2} - 1)/x$ и $f_{-1} = g_{-1}\bigl(x/\sqrt{2m}\bigr)$; тогда $f_{-1}(0) = 0$ и $r_{-1}(m)$ нетрудно получить из $\sum_{0\le t<m} f_{-1}(t)$. Равенство (36) справедливо теперь и при $k = -1$, а оставшийся интеграл нам хорошо знаком (см. упр. 43):

$$\frac{2}{\sqrt{2m}}\int_0^m f_{-1}(x)\,dx = 2\int_0^m \frac{e^{-x^2/2m}-1}{x}\,dx = \int_0^{m/2} \frac{e^{-y}-1}{y}\,dy$$
$$= \int_0^1 \frac{e^{-y}-1}{y}\,dy + \int_1^{m/2} \frac{e^{-y}}{y}\,dy - \ln\frac{m}{2}$$
$$= -\gamma - \ln m + \ln 2 + O(e^{-m/2}).$$

Теперь у нас достаточно формул и фактов для того, чтобы вывести, наконец, ответ, который, как показано в упр. 44, имеет вид

$$W_n = \tfrac{1}{2} m \ln m + \tfrac{1}{2}(\gamma + \ln 2)m - \tfrac{2}{3}\sqrt{2\pi m} + \tfrac{49}{36} + O(n^{-1/2}), \quad m = n+1. \tag{37}$$

Этим завершается анализ метода пузырька.

Чтобы проанализировать метод обменной поразрядной сортировки, необходимо знать асимптотическое поведение при $n \to \infty$ конечной суммы

$$U_n = \sum_{k\geq 2} \binom{n}{k} (-1)^k \frac{1}{2^{k-1}-1}. \tag{38}$$

Определить его оказывается гораздо сложнее, чем решить другие задачи об асимптотическом поведении, с которыми мы сталкивались до сих пор; элементарные методы разложения в степенные ряды, формула суммирования Эйлера и т. п. здесь бессильны. Следующий вывод был предложен Н. Г. де Брейном (N. G. de Bruijn).

Чтобы избавиться в формуле (38) от подавляющего влияния больших множителей $\binom{n}{k}(-1)^k$, начнем с того, что перепишем сумму в виде бесконечного ряда:

$$U_n = \sum_{k\geq 2} \binom{n}{k} (-1)^k \sum_{j\geq 1} \left(\frac{1}{2^{k-1}}\right)^j = \sum_{j\geq 1} \left(2^j(1-2^{-j})^n - 2^j + n\right). \tag{39}$$

Если положить $x = n/2^j$, то член ряда запишется в виде

$$2^j(1-2^{-j})^n - 2^j + n = \frac{n}{x}\left(\left(1-\frac{x}{n}\right)^n - 1 + x\right).$$

При $x \leq n^\epsilon$ имеем

$$\left(1-\frac{x}{n}\right)^n = \exp\left(n\ln\left(1-\frac{x}{n}\right)\right) = \exp\left(-x + x^2\, O(n^{-1})\right), \tag{40}$$

а это наводит на мысль о том, что следует аппроксимировать (39) рядом

$$T_n = \sum_{j\geq 1} \left(2^j e^{-n/2^j} - 2^j + n\right). \tag{41}$$

Чтобы подтвердить правомерность такой аппроксимации, рассмотрим разность $U_n - T_n = X_n + Y_n$, где

$$\begin{aligned}
X_n &= \sum_{\substack{j\geq 1\\ 2^j<n^{1-\epsilon}}} \left(2^j(1-2^{-j})^n - 2^j e^{-n/2^j}\right) && [\text{члены при } x > n^\epsilon] \\
&= \sum_{\substack{j\geq 1\\ 2^j<n^{1-\epsilon}}} O(ne^{-n/2^j}) && [\text{поскольку } 0 < 1-2^{-j} < e^{-2^{-j}}] \\
&= O(n\log n\, e^{-n^\epsilon}) && [\text{поскольку имеется } O(\log n) \text{ членов}]
\end{aligned}$$

и

$$\begin{aligned}
Y_n &= \sum_{\substack{j\geq 1\\ 2^j\geq n^{1-\epsilon}}} \left(2^j(1-2^{-j})^n - 2^j e^{-n/2^j}\right) && [\text{члены при } x \leq n^\epsilon] \\
&= \sum_{\substack{j\geq 1\\ 2^j\geq n^{1-\epsilon}}} \left(e^{-n/2^j}\frac{n}{2^j}\, O(1)\right). && [\text{вследствие (40)}]
\end{aligned}$$

Ниже будет показано, что эта последняя сумма есть $O(1)$; значит, $U_n - T_n = O(1)$ (см. упр. 47).

До сих пор никакие методы, которые действительно отличались бы от применявшихся ранее, еще не использовались, но для анализа ряда T_n потребуется новая идея, основанная на простых принципах теории функций комплексного переменного. Если x — произвольное положительное число, то

$$e^{-x} = \frac{1}{2\pi i}\int_{1/2-i\infty}^{1/2+i\infty} \Gamma(z)x^{-z}\,dz = \frac{1}{2\pi}\int_{-\infty}^{\infty} \Gamma(\tfrac{1}{2}+it)x^{-(1/2+it)}\,dt. \tag{42}$$

Для доказательства этого тождества рассмотрим путь интегрирования, показанный на рис. 20, (a), где N, N' и M велики. Значение интеграла вдоль этого контура равно сумме вычетов внутри контура, а именно

$$\sum_{0\le k<M} x^{-(-k)} \lim_{z\to -k}(z+k)\Gamma(z) = \sum_{0\le k<M} x^k\,\frac{(-1)^k}{k!}.$$

Интеграл по верхнему отрезку контура есть $O\bigl(\int_{-\infty}^{1/2}|\Gamma(t+iN)|\,x^{-t}\,dt\bigr)$, и имеется хорошо известная оценка

$$\Gamma(t+iN) = O\bigl(|t+iN|^{t-1/2}e^{-t-\pi N/2}\bigr) \qquad \text{при } N\to\infty.$$

[Свойства гамма-функций рассматриваются, например, в книге Erdélyi, Magnus, Oberhettinger and Tricomi, *Higher Transcendental Functions* 1 (New York: McGraw-Hill, 1953), Chapter 1.] Поэтому интегралом по верхнему отрезку можно пренебречь: $O\bigl(e^{-\pi N/2}\int_{-\infty}^{1/2}(N/xe)^t\,dt\bigr)$. Интеграл по нижнему отрезку ведет себя столь же безобидно. Для вычисления интеграла по левому отрезку контура воспользуемся тем фактом, что

$$\begin{aligned}\Gamma\bigl(\tfrac{1}{2}+it-M\bigr) &= \Gamma\bigl(\tfrac{1}{2}+it\bigr)/\bigl(-M+\tfrac{1}{2}+it\bigr)\ldots\bigl(-1+\tfrac{1}{2}+it\bigr)\\ &= \Gamma\bigl(\tfrac{1}{2}+it\bigr)O\bigl(1/(M-1)!\bigr).\end{aligned}$$

Следовательно, интеграл по левой стороне есть

$$O\bigl(x^{M-1/2}/(M-1)!\bigr)\int_{-\infty}^{\infty}\left|\Gamma\bigl(\frac{1}{2}+it\bigr)\right|\,dt.$$

Поэтому при M, N, $N'\to\infty$ уцелеет лишь интеграл по правой стороне; тем самым доказано тождество (42). В действительности тождество (42) остается в силе и в том случае, если заменить $\frac{1}{2}$ любым положительным числом.

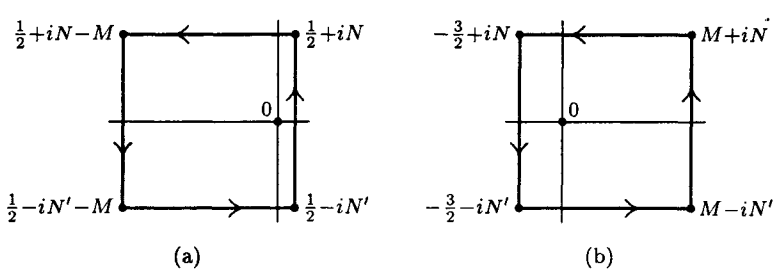


Рис. 20. Контуры интегрирования для тождеств с гамма-функциями.

Рассуждая аналогично, можно вывести и другие полезные соотношения, содержащие гамма-функции. Величину x^{-z} можно заменить другими функциями от z; можно также заменить другой величиной константу $\frac{1}{2}$. Например,

$$\frac{1}{2\pi i}\int_{-3/2-i\infty}^{-3/2+i\infty}\Gamma(z)x^{-z}\,dz = e^{-x}-1+x, \tag{43}$$

а это — критическая величина в формуле (41) для T_n:

$$T_n = n\sum_{j\ge1}\frac{1}{2\pi i}\int_{-3/2-i\infty}^{-3/2+i\infty}\Gamma(z)(n/2^j)^{-1-z}\,dz. \tag{44}$$

Суммирование можно внести под знак интеграла, так как сходимость здесь достаточно хорошая. Имеем

$$\sum_{j\ge1}(n/2^j)^w = n^w\sum_{j\ge1}(1/2^w)^j = n^w/(2^w-1), \qquad \text{если } \Re(w)>0,$$

поскольку $|2^w| = 2^{\Re(w)} > 1$. Поэтому

$$T_n = \frac{n}{2\pi i}\int_{-3/2-i\infty}^{-3/2+i\infty}\frac{\Gamma(z)n^{-1-z}}{2^{-1-z}-1}\,dz, \tag{45}$$

и остается оценить последний интеграл.

На этот раз интегрирование производится по контору, который больше вытянут *вправо*, как изображено на рис. 20, (b). Интеграл по верхнему отрезку есть $O(n^{1/2}e^{-\pi N/2}\int_{-3/2}^{M}|M+iN|^t\,dt)$, если $2^{iN}\ne1$, а интеграл по нижнему отрезку также пренебрежимо мал, когда N и N' значительно больше, чем M. Интеграл по правому отрезку равен $O(n^{-1-M}\int_{-\infty}^{\infty}|\Gamma(M+it)|\,dt)$. Зафиксировав M и устремив N, $N'\to\infty$, можно показать, что $-T_n/n$ есть $O(n^{-1-M})$ плюс сумма вычетов в области $-3/2<\Re(z)<M$. Коэффициент $\Gamma(z)$ имеет простые полюсы при $z=-1$ и $z=0$, в то время как n^{-1-z} не имеет полюсов, а $1/(2^{-1-z}-1)$ имеет простые полюсы при $z=-1+2\pi ik/\ln2$.

Наибольшую трудность представляет двойной полюс в точке $z=-1$. Если $w=z+1$ мало́, то можно воспользоваться известным соотношением

$$\Gamma(z+1) = \exp(-\gamma z+\zeta(2)z^2/2-\zeta(3)z^3/3+\zeta(4)z^4/4-\cdots),$$

где $\zeta(s)=1^{-s}+2^{-s}+3^{-s}+\cdots=H_\infty^{(s)}$, для вывода следующих разложений:

$$\Gamma(z) = \frac{\Gamma(w+1)}{w(w-1)} = -w^{-1}+(\gamma-1)+O(w),$$

$$n^{-1-z} = 1-w\ln n+O(w^2),$$

$$1/(2^{-1-z}-1) = -w^{-1}/\ln2-\tfrac12+O(w).$$

Вычет в точке $z=-1$ равен коэффициенту при w^{-1} в произведении этих трех формул, а именно $\frac12-(\ln n+\gamma-1)/\ln2$. Прибавляя остальные вычеты, получаем формулу

$$\frac{T_n}{n} = \frac{\ln n+\gamma-1}{\ln2}-\frac12+\delta(n)+\frac{2}{n}+O(n^{-M}) \tag{46}$$

для любого большого M, где $\delta(n)$ — функция довольно необычного вида:

$$\delta(n) = \frac{2}{\ln 2} \sum_{k\geq 1} \Re\bigl(\Gamma(-1 - 2\pi ik/\ln 2)\exp(2\pi ik\lg n)\bigr). \tag{47}$$

Заметим, что $\delta(n) = \delta(2n)$. Среднее значение $\delta(n)$ равно 0, так как среднее значение каждого слагаемого равно 0. (Можно считать, что величина $(\lg n) \bmod 1$ имеет равномерное распределение, принимая во внимание результаты, которые имеют отношение к числам с плавающей точкой, полученные в разделе 4.2.4.) Кроме того, поскольку $|\Gamma(-1+it)| = \bigl|\pi/(t(1+t^2)\sinh \pi t)\bigr|^{1/2}$, нетрудно показать, что

$$\bigl|\delta(n)\bigr| < 0.000000173. \tag{48}$$

Таким образом, в практических приложениях $\delta(n)$ можно спокойно отбросить. Что касается теории, то без $\delta(n)$ получить асимптотический ряд для U_n невозможно; именно поэтому анализ величины U_n довольно затруднителен.

Из определения T_n в (41) немедленно следует, что

$$\frac{T_{2n}}{2n} = \frac{T_n}{n} + 1 - \frac{1}{n} + \frac{e^{-n}}{n}. \tag{49}$$

Таким образом, слагаемым $O(n^{-M})$, представляющим ошибку, в выражении (46) нельзя пренебречь и заменить его нулем. Однако в упр. 54 предлагается другой подход к анализу, при котором удается избежать появления такого слагаемого, сформировав довольно необычный сходящийся ряд.

Итак, сумма (38) сведена к следующему выражению:

$$U_n = n\lg n + n\left(\frac{\gamma - 1}{\ln 2} - \frac{1}{2} + \delta(n)\right) + O(1). \tag{50}$$

Метод гамма-функций, который использовался для получения этого результата, представляет собой частный случай более общего метода *преобразования Меллина*, которое исключительно полезно для анализа рекуррентных методов, связанных с поразрядным представлением. Другие примеры применения этого метода гамма-функций можно найти в упр. 51–53 и в разделе 6.3. Прекрасным введением в преобразования Меллина и его приложения для анализа алгоритмов является работа P. Flajolet, X. Gourdon and P. Dumas, *Theoretical Computer Science* **144** (1995), 3–58.

УПРАЖНЕНИЯ

1. [*M20*] Пусть $a_1 \ldots a_n$ — перестановка множества $\{1, \ldots, n\}$ и пусть i и j таковы, что $i < j$ и $a_i > a_j$. Пусть $a'_1 \ldots a'_n$ — перестановка, которая получается из $a_1 \ldots a_n$, если поменять местами a_i и a_j. Может ли в $a'_1 \ldots a'_n$ быть больше инверсий, чем в $a_1 \ldots a_n$?

▸ **2.** [*M25*] (a) Каково минимальное число обменов, необходимых для того, чтобы рассортировать перестановку 3 7 6 9 8 1 4 5 2? (b) В общем случае пусть дана перестановка $\pi = a_1 \ldots a_n$ множества $\{1, \ldots, n\}$ и пусть $\mathrm{xch}(\pi)$ — минимальное число обменов записей, в результате которых перестановка π будет рассортирована в порядке возрастания. Выразите $\mathrm{xch}(\pi)$ через "более простые" характеристики перестановки π (см. упр. 5.1.4–41).

3. [*10*] Является ли устойчивой сортировка методом пузырька (алгоритм B)?

4. [*M23*] Если на шаге B4 получится $t = 1$, то на самом деле работу алгоритма B можно сразу же заканчивать, потому что на следующем шаге B2 не выполнится никаких полезных действий. Какова вероятность того, что при сортировке случайной перестановки на шаге B4 окажется $t = 1$?

5. [*M25*] Пусть $b_1\, b_2 \ldots b_n$ — таблица инверсий перестановки $a_1\, a_2 \ldots a_n$. Покажите, что после r проходов сортировки методом пузырька значение переменной BOUND будет равно $\max\{b_i + i \mid b_i \geq r\} - r$ при $0 \leq r \leq \max(b_1, \ldots, b_n)$.

6. [*M22*] Пусть $a_1 \ldots a_n$ — перестановка множества $\{1, \ldots, n\}$ и пусть $a'_1 \ldots a'_n$ — обратная к ней перестановка. Покажите, что число проходов, необходимых для того, чтобы рассортировать $a_1 \ldots a_n$ методом пузырька, равно $1 + \max(a'_1 - 1,\, a'_2 - 2, \ldots, a'_n - n)$.

7. [*M28*] Вычислите стандартное отклонение числа проходов при сортировке методом пузырька и выразите его через n и функцию $P(n)$. [Ср. с формулами (6) и (7).]

8. [*M24*] Выведите формулу (8).

9. [*M48*] Проанализируйте число проходов и число сравнений в алгоритме шейкер-сортировки. (*Замечание.* Полезная информация содержится в упр. 5.4.8–9.)

10. [*M26*] Пусть $a_1\, a_2 \ldots a_n$ — 2-упорядоченная перестановка множества $\{1, 2, \ldots, n\}$.

a) Каковы координаты конечных точек a_i-го шага соответствующего пути на решетке (см. рис. 11 на с. 106)?

b) Докажите, что сравнение и/или обмен элементов $a_1 : a_2$, $a_3 : a_4$, ... соответствует перегибанию пути относительно диагонали, как на рис. 18, (b).

c) Докажите, что сравнение и/или обмен элементов $a_2 : a_{2+d}$, $a_4 : a_{4+d}$, ... соответствует перегибанию пути относительно линии, расположенной на m единиц ниже диагонали, как на рис. 18, (c), (d) и (e), если $d = 2m - 1$.

▸ **11.** [*M25*] На какой перестановке множества $\{1, 2, \ldots, 16\}$ достигается максимум числа обменов записей в алгоритме Бэтчера?

12. [*24*] Напишите MIX-программу для алгоритма M, предполагая, что MIX — компьютер, в системе команд которого имеются команды AND и SRB. Сколько времени потребуется такой программе, чтобы рассортировать шестнадцать записей из табл. 1?

13. [*10*] Устойчива ли сортировка Бэтчера?

14. [*M21*] Пусть $c(N)$ — число сравнений ключей при сортировке N элементов методом Бэтчера; оно равно количеству выполнений шага M4.

a) Покажите, что $c(2^t) = 2c(2^{t-1}) + (t-1)2^{t-1} + 1$ при $t \geq 1$.

b) Найдите простое выражение для $c(2^t)$ как функцию от t. *Указание.* Рассмотрите последовательность $x_t = c(2^t)/2^t$.

15. [*M38*] Назначение этого упражнения — проанализировать функцию $c(N)$ из упр. 14 и вывести формулу для $c(N)$, если $N = 2^{e_1} + 2^{e_2} + \cdots + 2^{e_r}$, $e_1 > e_2 > \cdots > e_r \geq 0$.

a) Пусть $a(N) = c(N+1) - c(N)$. Докажите, что $a(2n) = a(n) + \lfloor \lg(2n) \rfloor$ и $a(2n+1) = a(n) + 1$; отсюда

$$a(N) = \binom{e_1 + 1}{2} - r(e_1 - 1) + (e_1 + e_2 + \cdots + e_r).$$

b) Пусть $x(n) = a(n) - a(\lfloor n/2 \rfloor)$, так что $a(n) = x(n) + x(\lfloor n/2 \rfloor) + x(\lfloor n/4 \rfloor) + \cdots$. Пусть $y(n) = x(1) + x(2) + \cdots + x(n)$ и $z(2n) = y(2n) - a(n)$, $z(2n+1) = y(2n+1)$. Докажите, что $c(N+1) = z(N) + 2z(\lfloor N/2 \rfloor) + 4z(\lfloor N/4 \rfloor) + \cdots$.

c) Докажите, что $y(N) = N + (\lfloor N/2 \rfloor + 1)(e_1 - 1) - 2^{e_1} + 2$.

d) Теперь соберите все вместе и найдите выражение $c(N)$ через показатели e_j при фиксированном значении r.

16. [*HM42*] Найдите значение асимптотического выражения для среднего числа операций обмена, когда алгоритм Бэтчера применяется к случайной перестановке N различных элементов, полагая, что N есть степень двойки.

▶ **17.** [*20*] Где в алгоритме Q используется тот факт, что K_0 и K_{N+1} имеют значения, постулированные неравенствами (13)?

▶ **18.** [*20*] Объясните, как работает алгоритм Q в случае, когда все ключи в исходном массиве равны. Что произойдет, если на шагах Q3 и Q5 заменить знаки "$<$" знаками "$\leq$"?

19. [*15*] Будет ли алгоритм Q по-прежнему работать правильно, если вместо стека (последним пришел — первым вышел) воспользоваться очередью (первым пришел — первым вышел)?

20. [*M20*] Выразите наибольшее число элементов, которые могут одновременно оказаться в стеке во время работы алгоритма Q, в виде функции от M и N.

21. [*20*] Объясните, почему для фазы разделения в алгоритме Q требуется именно столько сравнений и пересылок, сколько определено в (17).

22. [*M25*] Пусть p_{kN} — вероятность того, что величина A в (16) будет равна k, если алгоритм Q применяется к случайной перестановке множества $\{1, 2, \ldots, N\}$, и пусть $A_N(z) = \sum_k p_{kN} z^k$ — соответствующая производящая функция. Докажите, что $A_N(z) = 1$ при $N \leq M$, а $A_N(z) = z(\sum_{1 \leq s \leq N} A_{s-1}(z) A_{N-s}(z))/N$ при $N > M$. Найдите аналогичные рекуррентные соотношения, определяющие другие распределения вероятностей $B_N(z)$, $C_N(z)$, $D_N(z)$, $E_N(z)$, $S_N(z)$.

23. [*M23*] Пусть A_N, B_N, D_N, E_N, S_N — средние значения соответствующих величин в (16) при сортировке случайной перестановки множества $\{1, 2, \ldots, N\}$. Найдите для этих величин рекуррентные соотношения, аналогичные (18), затем найдите решения этих соотношений и получите формулы (25).

24. [*M21*] Очевидно, в алгоритме Q выполняется несколько больше сравнений, чем необходимо, потому что на шаге Q3 может получиться $i = j$, а на шаге Q5 — даже $i > j$. Сколько сравнений C_N выполнялось бы в среднем, если бы исключались все сравнения при $i \geq j$?

25. [*M20*] Чему равны точные значения величин A, B, C, D, E и S для программы Q, когда исходные ключи представляют собой упорядоченный набор чисел $1\,2 \ldots N$ в предположении, что $N > M$.

▶ **26.** [*M24*] Постройте исходную последовательность, при которой программа Q работала бы еще медленнее, чем в упр. 25. (Попытайтесь найти по-настоящему отвратительный случай.)

27. [*M28*] (Р. Седгевик (R. Sedgewick).) Рассмотрите *наилучший* случай для алгоритма Q. Найдите перестановку множества $\{1, 2, \ldots, 23\}$, которая требует меньше всего времени для сортировки при $N = 23$ и $M = 3$.

28. [*M26*] Найдите рекуррентное соотношение, аналогичное (20), которому удовлетворяет среднее число сравнений в алгоритме Q, модифицированном Синглтоном (т. е. когда в качестве s выбирается не $s = K_1$, а медиана из $\{K_1, K_{\lfloor (N+1)/2 \rfloor}, K_N\}$). Не обращайте внимания на сравнения, которые необходимы при вычислении медианы s.

29. [*HM40*] Найдите значение асимптотического выражения для числа сравнений в алгоритме Синглтона "медиана из трех" (это упражнение является продолжением упр. 28).

▶ **30.** [*25*] (П. Шеклтон (P. Shackleton).) При сортировке ключей длиной *несколько машинных слов* работа многих алгоритмов все более замедляется по мере упорядочения массива, поскольку для определения правильного лексикографического порядка равных или почти равных ключей необходимо сравнить несколько пар слов (см. упр. 5–5). В массивах, которые встречаются на практике, часто содержатся почти равные ключи, и это явление может заметно отразиться на времени сортировки.

Объясните, как можно усовершенствовать алгоритм Q, чтобы избежать этого затруднения. В подмассиве, о котором известно, что старшие k слов во всех ключах содержат постоянные значения, следует проверять лишь $(k+1)$-е слова ключей.

▶ **31.** [*20*] (Ч. Э. Р. Хоар (C. A. R. Hoare).) Предположим, что необходимо не рассортировать массив, а лишь определить m-й наименьший по величине элемент из заданного множества n элементов. Покажите, что алгоритм быстрой сортировки можно приспособить для этой цели, сократив значительную часть вычислений, которые используются для полной сортировки.

32. [*M40*] Найдите простое выражение в замкнутом виде для C_{nm} — среднего числа сравнений ключей, необходимых для поиска m-го наименьшего из n элементов по методу быстрого поиска (см. упр. 31). (Для простоты можете положить $M = 1$; т. е. предполагается не использовать специальную методику обработки коротких подмассивов.) Каково асимптотическое поведение параметра $C_{(2m-1)m}$ — среднего числа сравнений, необходимых для нахождения медианы из $2m-1$ элементов методом Хоара?

▶ **33.** [*15*] Разработайте алгоритм переразмещения чисел в некоторой заданной таблице таким образом, чтобы все *отрицательные* значения предшествовали положительным. Элементы полностью сортировать не нужно: достаточно просто отделить отрицательные числа от положительных. Алгоритм должен быть построен таким образом, чтобы минимизировалось количество операций обмена записей в памяти.

34. [*20*] Как можно ускорить циклы проверки разрядов при обменной поразрядной сортировке (шаги от R3 до R6)?

35. [*M23*] Проанализируйте статистические характеристики A, B, C, G, K, L, R, S и X, которые получаются при обменной поразрядной сортировке исходных данных наподобие (i).

36. [*M27*] Для любой данной последовательности чисел $\langle a_n\rangle = a_0, a_1, a_2, \ldots$ определите ее *биномиальное преобразование* $\langle \hat a_n\rangle = \hat a_0, \hat a_1, \hat a_2, \ldots$ с помощью правила

$$\hat a_n = \sum_k \binom{n}{k}(-1)^k a_k.$$

a) Докажите, что $\langle \hat{\hat a}_n\rangle = \langle a_n\rangle$.
b) Найдите биномиальные преобразования последовательностей $\langle 1\rangle$, $\langle n\rangle$ и $\langle \binom{n}{m}\rangle$ при фиксированном m, $\langle a^n\rangle$ — при фиксированном a, $\langle \binom{n}{m}a^n\rangle$ — при фиксированных a и m.
c) Предположим, что последовательность $\langle x_n\rangle$ удовлетворяет соотношению

$$x_n = a_n + 2^{1-n}\sum_{k\ge 2}\binom{n}{k}x_k \qquad \text{при } n\ge 2; \qquad x_0 = x_1 = a_0 = a_1 = 0.$$

Докажите, что решением этого рекуррентного соотношения будет

$$x_n = \sum_{k\ge 2}\binom{n}{k}(-1)^k\frac{2^{k-1}\hat a_k}{2^{k-1}-1} = a_n + \sum_{k\ge 2}\binom{n}{k}(-1)^k\frac{\hat a_k}{2^{k-1}-1}.$$

37. [*M28*] Определите все такие последовательности $\langle a_n\rangle$, что $\langle \hat a_n\rangle = \langle a_n\rangle$ в смысле упр. 36.

▶ **38.** [*M30*] Найдите A_N, B_N, C_N, G_N, K_N, L_N, R_N и X_N — средние значения величин в (29) в случае, когда поразрядной сортировке подвергаются исходные данные наподобие (ii). Выразите ответы через N и функции

$$U_n = \sum_{k\ge 2}\binom{n}{k}\frac{(-1)^k}{2^{k-1}-1}, \qquad V_n = \sum_{k\ge 2}\binom{n}{k}\frac{(-1)^k k}{2^{k-1}-1} = n(U_n - U_{n-1}).$$

[*Указание.* См. упр. 36.]

39. [*20*] Результаты (30) показывают, что поразрядная обменная сортировка, примененная к случайным входным данным, требует около $1.44N$ итераций. Докажите, что для быстрой сортировки никогда не требуется более N итераций, и объясните, почему при обменной поразрядной сортировке часто необходимо более N итераций.

40. [*21*] Объясните, как нужно изменить алгоритм R, чтобы он работал достаточно эффективно и в том случае, когда в сортируемых массивах содержится много равных ключей.

▶ **41.** [*30*] Разработайте такой способ обмена записей $R_l \ldots R_r$, который обеспечивает их разбиение на три блока: (i) $K_k < K$ при $1 \le k < i$; (ii) $K_k = K$ для $i \le k \le j$; (iii) $K_k > K$ для $j < k \le r$. Схематически окончательная компоновка должна иметь следующий вид:

$< K$	$= K$	$> K$

$l \qquad i \qquad j \qquad r$

42. [*HM32*] Докажите, что для любого действительного числа $c > 0$ вероятность того, что алгоритм Q потребует менее $(c+1)(N+1)H_N$ сравнений при сортировке данных, меньше e^{-c}. (Верхняя грань представляет особый интерес, если задать условие, скажем, $c = N^{\epsilon}$.)

43. [*HM21*] Докажите, что $\int_0^1 y^{-1}(e^{-y}-1)\,dy + \int_1^\infty y^{-1}e^{-y}\,dy = -\gamma$. [*Указание.* Рассмотрите $\lim_{a\to 0+} y^{a-1}$.]

44. [*HM24*] Выведите формулу (37), как предложено в тексте настоящего раздела.

45. [*HM20*] Объясните, почему при $x > 0$ справедливо равенство (43).

46. [*HM20*] Каково значение выражения $(1/2\pi i)\int_{a-i\infty}^{a+i\infty} \Gamma(z)n^{s-z}\,dz/(2^{s-z}-1)$ при условиях, что s — целое положительное число и $0 < a < s$?

47. [*HM21*] Докажите, что $\sum_{j\ge 1}(n/2^j)e^{-n/2^j}$ — ограниченная функция от n.

48. [*HM24*] Найдите асимптотическое значение величины V_n, определенной в упр. 38, при помощи методов, аналогичных тем, которые в тексте настоящего раздела использовались для анализа величины U_n, и получите члены до $O(1)$.

49. [*HM24*] Продолжите асимптотическую формулу (47) для U_n до членов порядка $O(n^{-1})$.

50. [*HM24*] Найдите асимптотическое значение функции

$$U_{mn} = \sum_{k\ge 2} \binom{n}{k} (-1)^k \frac{1}{m^{k-1}-1},$$

где m — произвольное фиксированное число, большее 1. (Эта функция при целом m, большем 2, потребуется для исследования общего случая обменной поразрядной сортировки, а также для анализа алгоритмов поиска в "памяти луча" в разделе 6.3.)

▶ **51.** [*HM28*] Покажите, что для вывода асимптотического разложения $r_k(m)$ можно воспользоваться методом исследования асимптотических задач при помощи гамма-функций вместо формулы суммирования Эйлера (см. (35)). (Это предоставляет единообразный способ анализа $r_k(m)$ при всех k без таких трюков, как введение в рассмотрение функции $g_{-1}(x) = (e^{-x^2}-1)/x$.)

52. [*HM35*] (Н. Г. де Брейн.) Каково асимптотическое поведение суммы

$$S_n = \sum_{t\ge 1} \binom{2n}{n+t} d(t),$$

где $d(t)$ — количество делителей числа t? (Таким образом, $d(1) = 1$, $d(2) = d(3) = 2$, $d(4) = 3$, $d(5) = 2$ и т. д. Этот вопрос возникает в связи с анализом алгоритма прохождения дерева из упр. 2.3.1–11.) Найдите значение величины $S_n/\binom{2n}{n}$ до членов порядка $O(n^{-1})$.

53. [*HM42*] Проанализируйте число проверок разрядов и число обменов, выполняемых при обменной поразрядной сортировке, если исходными данными служат двоичные числа с бесконечной точностью в диапазоне $[0\,..\,1)$, каждый разряд которых независимо принимает значение 1 с вероятностью p. (В основном тексте раздела обсуждался лишь случай, когда $p = \frac{1}{2}$; применявшиеся методы можно обобщить для произвольного p.) Рассмотрите особо случай, когда $p = 1/\phi = .61803\ldots$.

54. [*HM24*] (С. О. Райс (S. O. Rice).) Покажите, что U_n можно записать в виде

$$U_n = (-1)^n \frac{n!}{2\pi i} \oint_C \frac{dz}{z(z-1)\ldots(z-n)} \frac{1}{2^{z-1}-1},$$

где C — замкнутая кривая, охватывающая область около точек $2, 3, \ldots, n$. В результате замены C произвольно большой окружностью с центром в начале координат получаем сходящийся ряд

$$U_n = \frac{(H_{n-1}-1)n}{\ln 2} - \frac{n}{2} + 2 + \frac{2}{\ln 2} \sum_{m\geq 1} \Re\left(B(n+1, -1+ibm)\right),$$

где $b = 2\pi/\ln 2$ и $B(n+1, -1+ibm) = \Gamma(n+1)\Gamma(-1+ibm)/\Gamma(n+ibm) = n!/\prod_{k=0}^{n}(k-1+ibm)$.

▶ **55.** [*22*] Покажите, как нужно изменить программу Q, чтобы в качестве разделяющего элемента выбиралась медиана из трех ключей (28), полагая $M > 1$.

56. [*M43*] Проанализируйте поведение в среднем параметров, от которых зависит время работы программы Q, если программа изменена так, что она выбирает медиану из трех элементов, как в упр. 55 (см. упр. 29).

5.2.3. Сортировка посредством выбора

Еще одно важное семейство методов сортировки основано на идее многократного выбора. Вероятно, простейшая сортировка посредством выбора сводится к следующему.

i) Найти наименьший ключ, переслать соответствующую запись в область вывода и заменить ключ значением ∞ (которое по предположению больше любого реального ключа).

ii) Повторить шаг (i). На этот раз будет выбран ключ, наименьший из оставшихся, так как ранее наименьший ключ был заменен значением ∞.

iii) Повторять шаг (i) до тех пор, пока не будут выбраны N записей.

Заметим, что этот метод требует наличия всех исходных элементов до начала сортировки, а элементы вывода порождает последовательно, один за другим. Картина, по существу, противоположна картине, возникающей при использовании метода вставок, в котором исходные записи поступают последовательно, но вплоть до завершения сортировки об окончательном результате ничего неизвестно.

На шаге (i) требуется выполнить $N-1$ сравнений каждый раз, когда выбирается очередная запись. Также необходимо выделить в памяти отдельную область для накопления результата. Имеется очевидный способ несколько поправить ситуацию: выбранное значение записать в соответствующую окончательную позицию, а запись, которая ее занимала, перенести на место выбранной. Тогда эту позицию не нужно будет рассматривать вновь при последующих выборах и не придется иметь дело с ключом бесконечно большой величины. На этой идее основан наш первый алгоритм сортировки посредством выбора.

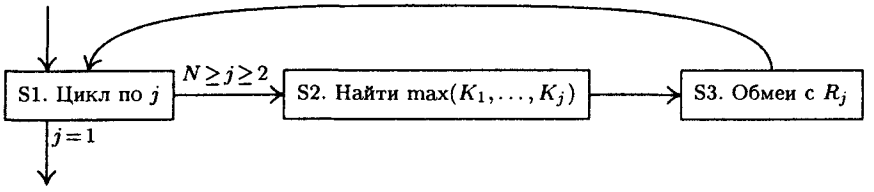


Рис. 21. Сортировка посредством простого выбора.

Алгоритм S (*Сортировка посредством простого выбора*). Записи $R_1, \ldots, R_N$ перекомпоновываются в пределах того же фрагмента памяти. После завершения сортировки их ключи будут упорядочены: $K_1 \leq \cdots \leq K_N$. Сортировка базируется на описанном выше методе, если не считать того, что более удобно, оказывается, сначала выбирать *наибольший* элемент, затем — второй по величине и т. д. (рис. 21).

S1. [Цикл по j.] Выполнить шаги S2 и S3 при $j = N, N-1, \ldots, 2$.

S2. [Найти $\max(K_1, \ldots, K_j)$.] Найти наибольший из ключей $K_j, K_{j-1}, \ldots, K_1$; пусть это будет K_i, где i выбирается как можно бóльшим.

S3. [Поменять местами с R_j.] Взаимно переставить записи $R_i \leftrightarrow R_j$. (Теперь записи $R_j, \ldots, R_N$ занимают свои окончательные позиции.) ▮

В табл. 1 продемонстрирован процесс выполнения этого алгоритма при обработке шестнадцати ключей, выбранных нами для примеров. Элементы, претендующие на то, чтобы быть максимальными во время поиска на шаге S2, выделены полужирным шрифтом.

Таблица 1

СОРТИРОВКА ПОСРЕДСТВОМ ПРОСТОГО ВЫБОРА

503	087	512	061	**908**	170	**897**	275	653	426	154	509	612	677	**765**	**703**|
503	087	512	061	703	170	**897**	275	653	426	154	509	612	677	**765**|	908
503	087	512	061	703	170	**765**	275	653	426	154	509	612	**677**|	897	908
503	087	512	061	**703**	170	**677**	275	**653**	426	154	509	**612**|	765	897	908
503	087	512	061	612	170	**677**	275	**653**	426	154	**509**|	703	765	897	908
503	087	512	061	612	170	509	275	**653**	**426**	**154**|	677	703	765	897	908
...															
061|	087	154	170	275	426	503	509	512	612	653	677	703	765	897	908

Соответствующая MIX-программа довольно проста.

Программа S (*Сортировка посредством простого выбора*). Как и в предыдущих программах из этой главы, записи, находящиеся в ячейках от INPUT+1 до INPUT+N, сортируются в пределах того же фрагмента памяти по ключу, который занимает полное слово. Значения регистров таковы: rA $\equiv$ текущий максимум, rI1 $\equiv j-1$, rI2 $\equiv k$ (индекс при поиске), rI3 $\equiv i$. Предполагается, что $N \geq 2$.

```
01  START ENT1 N-1        1      S1. Цикл по j. j ← N.
02  2H    ENT2 0,1        N-1    S2. Найти max(K_1,...,K_j). k ← j-1.
03        ENT3 1,1        N-1    i ← j.
04        LDA  INPUT,3    N-1    rA ← K_i.
```

05	8H	CMPA	INPUT,2	A	
06		JGE	*+3	A	Переход, если $K_i \geq K_k$.
07		ENT3	0,2	B	Иначе — установить *igetsk*,
08		LDA	INPUT,3	B	rA $\leftarrow K_i$.
09		DEC2	1	A	$k \leftarrow k-1$.
10		J2P	8B	A	Повторить, если $k > 0$.
11		LDX	INPUT+1,1	$N-1$	*S3. Поменять местами с R_j.*
12		STX	INPUT,3	$N-1$	$R_i \leftarrow R_j$.
13		STA	INPUT+1,1	$N-1$	$R_j \leftarrow$ rA.
14		DEC1	1	$N-1$	
15		J1P	2B	$N-1$	$N \geq j \geq 2$. ▮

Время работы этой программы зависит от числа элементов N, числа сравнений A и числа правосторонних максимумов B. Нетрудно видеть, что независимо от значений исходных ключей

$$A = \binom{N}{2} = \frac{1}{2}N(N-1). \tag{1}$$

Следовательно, переменной является только величина B. Несмотря на всю безыскусность простого выбора, не так-то легко выполнить точный анализ величины B. В упр. 3–6 показано, что

$$B = \bigl(\min\ 0,\ \mathrm{ave}\,(N+1)H_N - 2N,\ \max\ \lfloor N^2/4 \rfloor\bigr). \tag{2}$$

В этом случае особенно интересным оказывается максимальное значение. Стандартное отклонение величины B имеет порядок $N^{3/4}$ (см. упр. 7).

Таким образом, среднее время работы программы S равно $2.5N^2 + 3(N+1)H_N + 3.5N - 11$ машинных циклов, т. е. данная программа работает лишь немногим медленнее программы, реализующей метод простых вставок (программа 5.2.1S). Интересно сравнить алгоритм S с сортировкой методом пузырька (алгоритм 5.2.2B), поскольку метод пузырька можно рассматривать как алгоритм выбора, в котором за один раз иногда выбирается более одного элемента. По этой причине при сортировке методом пузырька выполняется меньше сравнений, чем при простом выборе, и она, как может показаться, предпочтительнее. Но в действительности программа 5.2.2B работает более чем вдвое медленнее программы S! Сортировка методом пузырька проигрывает из-за того, что выполняется слишком много обменов, в то время как при сортировке посредством простого выбора пересылается очень мало данных.

Усовершенствования простого выбора. Существует ли какой-нибудь способ улучшения метода выбора, используемого в алгоритме S? Возьмем, к примеру, поиск максимума на шаге S2: возможен ли существенно более быстрый способ нахождения максимума? Ответ на этот вопрос — *нет*!

Лемма M. *В любом алгоритме нахождения максимума среди n элементов, основанном на сравнении пар элементов, необходимо выполнить, по крайней мере, $n-1$ сравнений.*

Доказательство. Если выполнено менее $n-1$ сравнений, то найдутся хотя бы два элемента, для которых не было обнаружено ни одного элемента, превосходящего их по величине. Следовательно, мы так и не узнаем, какой из этих двух элементов больше, а значит, не сможем определить максимум. ▮

Таким образом, процесс выбора, в котором выполняется поиск наибольшего элемента, должен включать не менее $n-1$ сравнений. Означает ли это, что для всех методов сортировки, основанных на n повторных выборах, число операций неизбежно будет порядка $\Omega(n^2)$? К счастью, лемма M применима только к *первому* шагу выбора; в дальнейшем можно использовать извлеченную ранее информацию. Например, в упр. 8 и 9 показано, что сравнительно простое изменение алгоритма S позволяет наполовину сократить среднее число сравнений.

Рассмотрим 16 чисел, представленных в табл. 1. Один из способов сэкономить время при последующих выборах — разбить все числа на четыре группы по четыре числа. Начать можно с определения наибольшего элемента каждой группы, а именно — с ключей

$$512,\ 908,\ 653,\ 765.$$

Тогда наибольший из этих четырех элементов (элемент 908) и будет наибольшим во всей последовательности. Чтобы получить второй по величине элемент, достаточно просмотреть 512, 653, 765 и остальные три элемента группы, содержащей 908; наибольший из $\{170, 897, 275\}$ равен 897, и тогда наибольшим среди элементов

$$512,\ 897,\ 653,\ 765$$

является 897. Аналогично для того, чтобы получить третий по величине элемент, определяем наибольший из $\{170, 275\}$, а затем наибольший из элементов

$$512,\ 275,\ 653,\ 765$$

и т. д. Каждый выбор, кроме первого, требует не более 6 дополнительных сравнений. В общем случае, если N — точный квадрат, можно разделить массив на $\sqrt{N}$ групп по $\sqrt{N}$ элементов. Любой выбор, кроме первого, требует не более чем $\sqrt{N}-2$ сравнений внутри группы ранее выбранного элемента плюс $\sqrt{N}-1$ сравнений среди "лидеров групп". Этот метод получил название *квадратичный выбор*; общее время его работы составляет порядка $O(N\sqrt{N})$, что существенно лучше, чем N^2.

Метод квадратичного выбора впервые был опубликован в работе E. H. Friend, *JACM* **3** (1956), 152–154. Э. Г. Френд указал, что его можно обобщить и получить методы кубической, четвертой и более высоких степеней выбора. Например, метод кубического выбора состоит в том, чтобы разделить массив на $\sqrt[3]{N}$ больших групп, в каждой из которых содержится по $\sqrt[3]{N}$ малых групп по $\sqrt[3]{N}$ записей. Время работы будет пропорционально $N\sqrt[3]{N}$. Если развить эту идею, можно прийти к тому, что Френд назвал "выбор n-й степени", базирующийся на структуре бинарного дерева. Время выполнения сортировки по этому методу пропорционально $N\log N$; будем называть его *выбором из дерева*.

Выбор из дерева. Принципы сортировки посредством выбора из дерева будет легче понять, если воспользоваться аналогией с типичным "турниром с выбыванием". Рассмотрим, например, результаты соревнования по настольному теннису, показанные на рис. 22. В первом туре Ким побеждает Сэнди, а Крис побеждает Лу; затем в следующем туре Крис выигрывает у Кима и т. д.

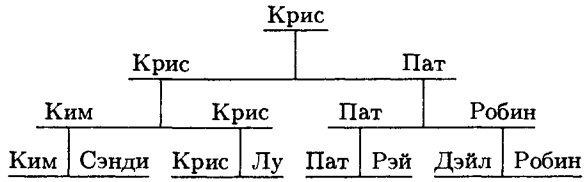


Рис. 22. Турнир по настольному теннису.

На рис. 22 показано, что Крис — чемпион среди восьми участников. Для того чтобы определить это, потребовалось $8 - 1 = 7$ матчей (т. е. сравнений). Пат вовсе необязательно будет вторым по силе игроком: любой из спортсменов, у которых выиграл Крис, включая даже проигравшего в первом туре Лу, может оказаться вторым по силе. Второго игрока можно определить, заставив Лу сыграть с Кимом, а победителя этого матча — с Патом. Всего двух дополнительных матчей достаточно для определения второго по силе игрока, исходя из соотношения сил, которое было учтено на основании предыдущих игр.

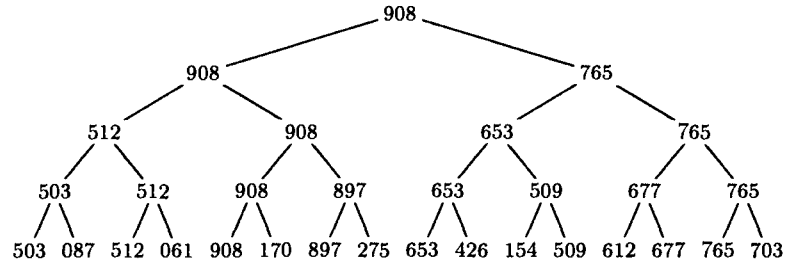
Вообще говоря, можно "вывести из турнира" игрока, находящегося в корне дерева, заменить его заведомо слабейшим новичком и повторить розыгрыш. Включение этого слабака приведет к тому, что первоначально второй по силе участник станет теперь наилучшим и именно он окажется в корне, если вновь вычислить победителей в верхних уровнях дерева. Для этого нужно изменить лишь один путь в дереве, так что для выбора следующего по силе игрока необходимо менее $\lceil \lg N \rceil$ дополнительных сравнений. Суммарное время выполнения такой сортировки посредством выбора примерно пропорционально N log N, как и утверждалось выше.

На рис. 23 показано, как применить эту схему к нашим 16 числам. Заметим, что для того, чтобы знать, куда вставлять следующий элемент $-\infty$, необходимо помнить, где находился ключ, оказавшийся в корне. Поэтому узлы разветвления в действительности содержат указатели или индексы, описывающие позицию ключа, а не сам ключ. Отсюда следует, что необходима память для N исходных записей, $N - 1$ указателей и N выводимых записей. (Разумеется, если вывод идет на ленту или диск, то не нужно сохранять выводимые записи в оперативной памяти.)

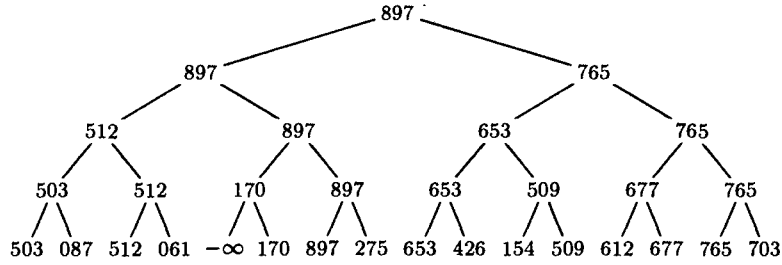
Чтобы оценить те замечательные усовершенствования, которые мы собираемся обсудить, в этом месте рекомендуется прервать чтение и выполнить упр. 10. Не усвоив базовые принципы рассматриваемого метода, нельзя двигаться дальше.

Одна из модификаций выбора из дерева, введенная, по существу, К. Э. Айверсоном (K. E. Iverson) [A Programming Language (Wiley, 1962), 223–227], устраняет необходимость указателей. Достигается это тем, что мы "заглядываем вперед": когда победитель матча на нижнем уровне поднимается вверх, на нижнем уровне его сразу же можно заменить элементом $-\infty$; когда же победитель перемещается вверх с одного разветвления на другое, его можно заменить игроком, который, в конце концов, все равно должен подняться на его прежнее место (а именно — наибольшим из двух ключей, расположенных под ним). Повторяя эту операцию как можно чаще, придем от рис. 23, (а) к рис. 24.

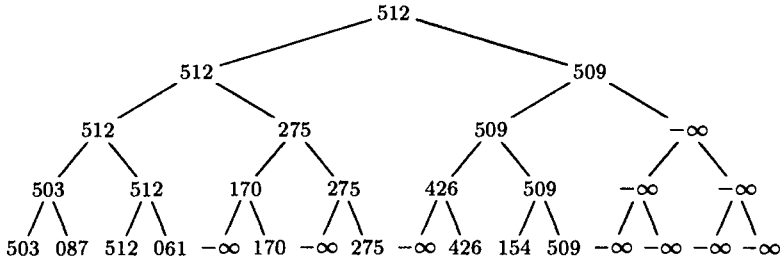
Коль скоро дерево построено таким образом, можно продолжать сортировку "нисходящим", а не "восходящим" методом, показанным на рис. 23: выводится



(a) Исходная конфигурация



(b) Ключ 908 заменен значением $-\infty$, а вторая по старшинству запись перемещается в корень



(c) Конфигурация после вывода 908, 897, 765, 703, 677, 653 и 612

Рис. 23. Пример сортировки посредством выбора.

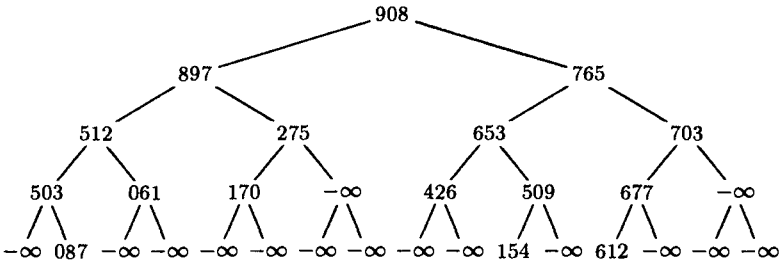


Рис. 24. Принцип Питера, примененный к сортировке. Каждый поднимается на свой уровень компетенции в иерархии.

элемент, находящийся в корне, перемещается вверх наибольший из его потомков, перемещается вверх наибольший из потомков последнего и т. д. Процесс начинает походить не столько на турнир по настольному теннису, сколько на систему выдвижений в корпорации.

Читатель должен уяснить, что у нисходящего метода есть важное достоинство — он позволяет избежать лишних сравнений $-\infty$ с $-\infty$. (Пользуясь восходящим методом, мы на более поздних стадиях сортировки всюду натыкаемся на $-\infty$, а при нисходящем методе можно на каждой стадии заканчивать преобразование дерева сразу же после занесения $-\infty$.)

На рис. 23 и 24 изображены *полные бинарные деревья* с 16 концевыми узлами (см. раздел 2.3.4.5). Такие деревья удобно хранить в последовательных ячейках памяти, как показано на рис. 25. Заметим, что родителем узла номер k является узел $\lfloor k/2 \rfloor$, а его потомками являются узлы $2k$ и $2k+1$. Отсюда вытекает еще одно преимущество нисходящего метода — зачастую значительно проще продвигаться вниз от узла k к узлам $2k$ и $2k+1$, чем вверх от узла k к узлам $k \oplus 1$ и $\lfloor k/2 \rfloor$. (Здесь через $k \oplus 1$ обозначено число $k+1$ или $k-1$ в зависимости от того, каким является k: четным или нечетным.)

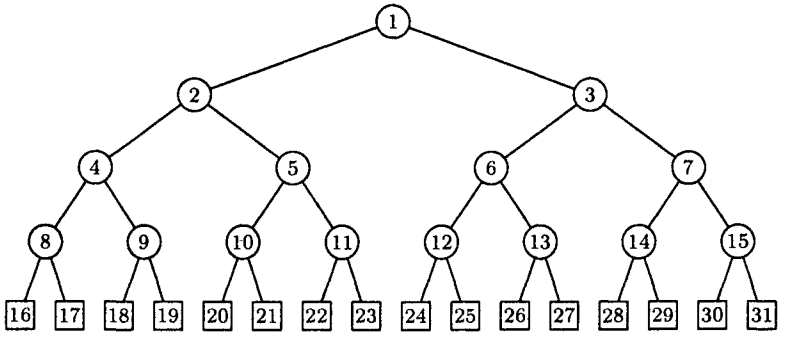


Рис. 25. Последовательное распределение памяти для полного бинарного дерева.

До сих пор в примерах выбора из дерева в той или иной мере предполагалось, что N есть степень 2. В действительности можно работать с произвольным значением N, так как полное бинарное дерево с N концевыми узлами нетрудно построить для любого N.

Мы подошли теперь к основному вопросу: нельзя ли в нисходящем методе обойтись совсем без $-\infty$? Не правда ли, было бы чудесно, если бы всю существенную информацию, которая представлена на рис. 24, удалось расположить в ячейках от 1 до 16 полного бинарного дерева безо всяких бесполезных "дыр", содержащих $-\infty$? Поразмыслив, можно прийти к выводу, что эта цель в действительности достижима, причем не только исключается $-\infty$, но и появляется возможность сортировать записи в пределах того же фрагмента памяти без дополнительной области для накопления результата. Это позволяет получить еще один важный алгоритм сортировки — пирамидальную сортировку (heap-sort). Его автор — Дж. У. Дж. Уильямс (J. W. J. Williams) [*CACM* **7** (1964), 347–348].

Пирамидальная сортировка. Будем называть массив ключей $K_1, K_2, \ldots, K_N$ *пирамидой*, если

$$K_{\lfloor j/2 \rfloor} \geq K_j \qquad \text{при } 1 \leq \lfloor j/2 \rfloor < j \leq N. \tag{3}$$

В этом случае $K_1 \geq K_2$, $K_1 \geq K_3$, $K_2 \geq K_4$ и т. д. Именно это условие выполняется на рис. 24. Из него следует, в частности, что наибольший ключ оказывается "на вершине пирамиды":

$$K_1 = \max(K_1, K_2, \ldots, K_N). \tag{4}$$

Если как-нибудь преобразовать произвольный исходный массив в пирамиду, то для получения эффективного алгоритма сортировки можно воспользоваться "нисходящей" процедурой выбора, подобной той, которая описана выше. Эффективный подход к построению пирамиды предложил Р. У. Флойд [R. W. Floyd, *CACM* **7** (1964), 701]. Предположим, удалось расположить массив таким образом, что

$$K_{\lfloor j/2 \rfloor} \geq K_j \qquad \text{при } l < \lfloor j/2 \rfloor < j \leq N, \tag{5}$$

где l — некоторое число ≥ 1. (В исходном массиве это условие выполняется автоматически для $l = \lfloor N/2 \rfloor$, поскольку ни один индекс j не удовлетворяет условию $\lfloor N/2 \rfloor < \lfloor j/2 \rfloor < j \leq N$.) Нетрудно понять, как, изменяя лишь поддерево с корнем в узле l, преобразовать массив, чтобы распространить неравенства (5) и на случай, когда $l = \lfloor j/2 \rfloor$. Следовательно, можно уменьшать l на единицу, пока, в конце концов, не будет достигнуто условие (3). Эти идеи Уильямса и Флойда приводят к изящному алгоритму, который заслуживает пристального внимания (рис. 26).

Алгоритм H (*Пирамидальная сортировка*). Записи $R_1, \ldots, R_N$ перекомпоновываются в пределах того же фрагмента памяти; после завершения сортировки их ключи будут упорядочены следующим образом: $K_1 \leq \cdots \leq K_N$. Сначала массив перестраивается в пирамиду, а затем вершина пирамиды многократно исключается и записывается на свое окончательное место. Предполагается, что $N \geq 2$.

H1. [Начальная установка.] Установить $l \leftarrow \lfloor N/2 \rfloor + 1$, $r \leftarrow N$.

H2. [Уменьшить l или r.] Если $l > 1$, установить $l \leftarrow l - 1$, $R \leftarrow R_l$, $K \leftarrow K_l$. (Если $l > 1$, значит, происходит преобразование исходного массива в пирамиду; если же $l = 1$, значит, ключи $K_1 K_2 \ldots K_r$ уже образуют пирамиду.) В противном случае установить $R \leftarrow R_r$, $K \leftarrow K_r$, $R_r \leftarrow R_1$ и $r \leftarrow r - 1$; если в результате оказалось, что $r = 1$, установить $R_1 \leftarrow R$ и завершить выполнение алгоритма.

H3. [Приготовиться к "протаскиванию".] Установить $j \leftarrow l$. (К этому моменту

$$K_{\lfloor k/2 \rfloor} \geq K_k \qquad \text{при } l < \lfloor k/2 \rfloor < k \leq r, \tag{6}$$

а запись R_k, $r < k \leq N$, занимает свое окончательное место. Шаги H3–H8 называются *алгоритмом "протаскивания"*; выполняемые при этом операции эквивалентны установке $R_l \leftarrow R$ с последующей перекомпоновкой записей $R_l, \ldots, R_r$ таким образом, чтобы условие (6) выполнялось и при $l = \lfloor k/2 \rfloor$.)

H4. [Продвинуться вниз.] Установить $i \leftarrow j$ и $j \leftarrow 2j$. (На последующих шагах $i = \lfloor j/2 \rfloor$.) Если $j < r$, перейти к шагу H5; если $j = r$, перейти к шагу H6; если же $j > r$, перейти к шагу H8.

H5. [Найти большего потомка.] Если $K_j < K_{j+1}$, то установить $j \leftarrow j + 1$.

H6. [Больше, чем K?] Если $K \geq K_j$, перейти к шагу H8.

H7. [Поднять его вверх.] Установить $R_i \leftarrow R_j$ и возвратиться к шагу H4.

H8. [Занести R.] Установить $R_i \leftarrow R$. (На этом алгоритм "протаскивания", начатый на шаге H3, заканчивается.) Возвратиться к шагу H2. ▌

Пирамидальную сортировку иногда описывают как Δ-алгоритм. Это обозначение указывает на характер изменения переменных l и r. Верхний треугольник соответствует фазе построения пирамиды, когда $r = N$, а l убывает до 1; нижний треугольник представляет фазу выбора, когда $l = 1$, а r убывает до 1. В табл. 2 показана пирамидальная сортировка все тех же 16 чисел. (В каждой строке изображено состояние в начале шага H2, скобки указывают на значения переменных l и r.)

Программа H (*Пирамидальная сортировка*). Записи, находящиеся в ячейках от INPUT+1 по INPUT+N, сортируются при помощи алгоритма H. В регистрах записана следующая информация: rI1 $\equiv l - 1$, rI2 $\equiv r - 1$, rI3 $\equiv i$, rI4 $\equiv j$, rI5 $\equiv r - j$, rA $\equiv K \equiv R$, rX $\equiv R_j$.

01	START	ENT1	N/2	1	*H1. Начальная установка.* $l \leftarrow \lfloor N/2 \rfloor + 1$.
02		ENT2	N-1	1	$r \leftarrow N$.
03	1H	DEC1	1	$\lfloor N/2 \rfloor$	$l \leftarrow l - 1$.
04		LDA	INPUT+1,1	$\lfloor N/2 \rfloor$	$R \leftarrow R_l$, $K \leftarrow K_l$.
05	3H	ENT4	1,1	P	*H3. Приготовиться к "протаскиванию".* $j \leftarrow l$.
06		ENT5	0,2	P	
07		DEC5	0,1	P	rI5 $\leftarrow r - j$.
08		JMP	4F	P	Перейти к шагу H4.
09	5H	LDX	INPUT,4	$B + A - D$	*H5. Найти большего потомка.*
10		CMPX	INPUT+1,4	$B + A - D$	
11		JGE	6F	$B + A - D$	Переход, если $K_j \geq K_{j+1}$.
12		INC4	1	C	В противном случае установить $j \leftarrow j + 1$.
13		DEC5	1	C	
14	9H	LDX	INPUT,4	$C + D$	rX $\leftarrow R_j$.
15	6H	CMPA	INPUT,4	$B + A$	*H6. Больше, чем K?*
16		JGE	8F	$B + A$	Перейти к шагу H8, если $K \geq K_j$.
17	7H	STX	INPUT,3	B	*H7. Поднять его вверх.* $R_i \leftarrow R_j$.
18	4H	ENT3	0,4	$B + P$	*H4. Продвинуться вниз.* $i \leftarrow j$.
19		DEC5	0,4	$B + P$	rI5 $\leftarrow$ rI5 $- j$.
20		INC4	0,4	$B + P$	$j \leftarrow j + j$.
21		J5P	5B	$B + P$	Перейти к шагу H5, если $j < r$.
22		J5Z	9B	$P - A + D$	Перейти к шагу H6, если $j = r$.
23	8H	STA	INPUT,3	P	*H8. Занести R.* $R_i \leftarrow R$.
24	2H	J1P	1B	P	*H2. Уменьшить l или r.*
25		LDA	INPUT+1,2	$N - 1$	Если $l = 1$, установить $R \leftarrow R_r$, $K \leftarrow K_r$.
26		LDX	INPUT+1	$N - 1$	
27		STX	INPUT+1,2	$N - 1$	$R_r \leftarrow R_1$.
28		DEC2	1	$N - 1$	$r \leftarrow r - 1$.
29		J2P	3B	$N - 1$	Перейти к шагу H3, если $r > 1$.
30		STA	INPUT+1	1	$R_1 \leftarrow R$. ▌

Эта программа приблизительно лишь вдвое длиннее программы S, но при больших N она гораздо более эффективна. Время выполнения программы зависит от следующих параметров:

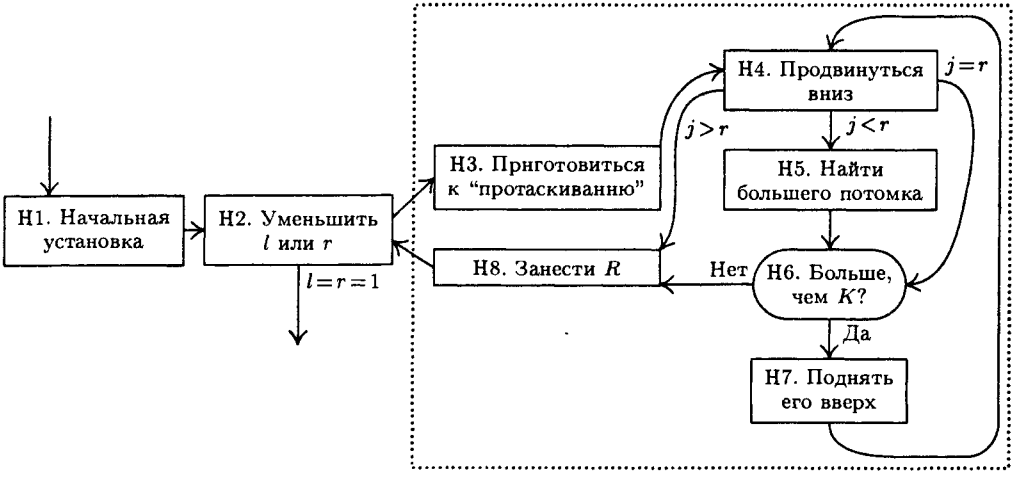


Рис. 26. Пирамидальная сортировка; пунктиром обозначен алгоритм "протаскивания".

Таблица 2

ПРИМЕР ПИРАМИДАЛЬНОЙ СОРТИРОВКИ

K_1	K_2	K_3	K_4	K_5	K_6	K_7	K_8	K_9	K_{10}	K_{11}	K_{12}	K_{13}	K_{14}	K_{15}	K_{16}	l	r
503	087	512	061	908	170	897	275	[653	426	154	509	612	677	765	703]	9	16
503	087	512	061	908	170	897	[703	653	426	154	509	612	677	765	275]	8	16
503	087	512	061	908	170	[897	703	653	426	154	509	612	677	765	275]	7	16
503	087	512	061	908	[612	897	703	653	426	154	509	170	677	765	275]	6	16
503	087	512	061	[908	612	897	703	653	426	154	509	170	677	765	275]	5	16
503	087	512	[703	908	612	897	275	653	426	154	509	170	677	765	061]	4	16
503	087	[897	703	908	612	765	275	653	426	154	509	170	677	512	061]	3	16
503	[908	897	703	426	612	765	275	653	087	154	509	170	677	512	061]	2	16
[908	703	897	653	426	612	765	275	503	087	154	509	170	677	512	061]	1	16
[897	703	765	653	426	612	677	275	503	087	154	509	170	061	512]	908	1	15
[765	703	677	653	426	612	512	275	503	087	154	509	170	061]	897	908	1	14
[703	653	677	503	426	612	512	275	061	087	154	509	170]	765	897	908	1	13
[677	653	612	503	426	509	512	275	061	087	154	170]	703	765	897	908	1	12
[653	503	612	275	426	509	512	170	061	087	154]	677	703	765	897	908	1	11
[612	503	512	275	426	509	154	170	061	087]	653	677	703	765	897	908	1	10
[512	503	509	275	426	087	154	170	061]	612	653	677	703	765	897	908	1	9
[509	503	154	275	426	087	061	170]	512	612	653	677	703	765	897	908	1	8
[503	426	154	275	170	087	061]	509	512	612	653	677	703	765	897	908	1	7
[426	275	154	061	170	087]	503	509	512	612	653	677	703	765	897	908	1	6
[275	170	154	061	087]	426	503	509	512	612	653	677	703	765	897	908	1	5
[170	087	154	061]	275	426	503	509	512	612	653	677	703	765	897	908	1	4
[154	087	061]	170	275	426	503	509	512	612	653	677	703	765	897	908	1	3
[087	061]	154	170	275	426	503	509	512	612	653	677	703	765	897	908	1	2

$P = N + \lfloor N/2 \rfloor - 2$ — число проходов с "протаскиванием";

A — число "протаскиваний", при которых ключ K, в конце концов, попадает во внутренний узел пирамиды;

B — суммарное число ключей, просмотренных во время "протаскиваний";

C — число присваиваний $j \leftarrow j + 1$ на шаге H5;

D — число случаев, когда $j = r$ на шаге H4.

Эти параметры проанализированы ниже; как показывают эксперименты, они лишь незначительно отклоняются от средних значений:

$$\begin{aligned} A &\approx 0.349N, & B &\approx N \lg N - 1.87N, \\ C &\approx \tfrac{1}{2} N \lg N - 0.94N, & D &\approx \ln N. \end{aligned} \tag{7}$$

При $N = 1000$, например, четыре эксперимента со случайными исходными данными показали соответственно результаты $A = 371, 351, 341, 340$, $B = 8055, 8072, 8094, 8108$, $C = 4056, 4087, 4017, 4083$ и $D = 12, 14, 8, 13$. Общее время выполнения программы $7A + 14B + 4C + 20N - 2D + 15\lfloor N/2 \rfloor - 28$ равно, таким образом, в среднем примерно $16N \lg N + 0.01N$ машинных циклов. Глядя на табл. 2, трудно поверить в то, что пирамидальная сортировка так уж эффективна: большие ключи перемещаются влево, прежде чем мы успеваем отложить их вправо! Это и в самом деле странный способ сортировки при малых N. Время сортировки 16 ключей из табл. 2 равно $1068u$, тогда как при обычным методе простых вставок (программа 5.2.1S) требуется всего $514u$. Сортировка посредством простого выбора (программа S) отнимает $853u$.

При больших N программа H более эффективна. Напрашивается сравнение с сортировкой методом Шелла с убывающим смещением (программа 5.2.1D) и быстрой сортировкой Хоара (программа 5.2.2Q), так как во всех трех программах сортировка выполняется путем сравнения ключей, причем требуется довольно малый объем дополнительной памяти или она не используется вовсе. При $N = 1000$ значения среднего времени выполнения равны приблизительно

$160000u$ для пирамидальной сортировки,

$130000u$ для сортировки методом Шелла,

$80000u$ для быстрой сортировки.

(MIX — типичный представитель большинства компьютеров, но, разумеется, на конкретных машинах получатся несколько иные относительные величины.) С ростом N пирамидальная сортировка превзойдет по скорости метод Шелла, но асимптотическое выражение для времени выполнения $16N \lg N \approx 23.08N \ln N$ никогда не станет лучше выражения для быстрой сортировки $11.67N \ln N$. Модификация пирамидальной сортировки, которая будет проанализирована в упр. 18, ускорит процесс вследствие снижения числа сравнений, но даже с этим усовершенствованием пирамидальная сортировка проигрывает по сравнению с быстрой сортировкой.

С другой стороны, быстрая сортировка эффективна лишь в среднем; в наихудшем случае ее время работы пропорционально N^2. Пирамидальная же сортировка обладает тем интересным свойством, что для нее наихудший случай не намного хуже среднего. Всегда выполняются неравенства

$$A \le 1.5N, \qquad B \le N\lfloor \lg N \rfloor, \qquad C \le N\lfloor \lg N \rfloor. \tag{8}$$

Таким образом, независимо от распределения исходных данных выполнение программы H не займет более $18N\lfloor \lg N \rfloor + 38N$ машинных циклов. Пирамидальная сортировка — первый из рассмотренных нами до сих пор методов сортировки, время работы которого *заведомо* имеет порядок $N \log N$. Сортировка посредством слияний, которая будет обсуждаться ниже, в разделе 5.2.4, тоже обладает таким свойством, но этот метод требует больше памяти.

Наибольший из включенных первым исключается. В главе 2 было показано, что линейные списки часто целесообразно классифицировать по характеру выполнения операций включения и исключения. *Стек* ведет себя по принципу "последним пришел — первым вышел" в том смысле, что при каждом исключении удаляется самый младший элемент списка (элемент, который был вставлен позже всех других элементов, присутствующих в данный момент в списке). Простая *очередь* ведет себя по принципу "первым пришел — первым вышел" в том смысле, что при каждом исключении удаляется самый старший из имеющихся элементов. В более сложных ситуациях, в таких как пример с моделированием лифта из раздела 2.2.5, необходим список наподобие "наименьший из включенных первым исключается", в котором при каждом исключении удаляется элемент, имеющий наименьший ключ. Такой список можно назвать *приоритетной очередью*, поскольку ключ каждого элемента отражает его относительную способность быстро покидать список. Сортировка посредством выбора — частный случай приоритетной очереди, над которой производится сначала N операций вставки, а затем N операций удаления.

Приоритетные очереди возникают в самых разнообразных приложениях. Например, в некоторых численных итеративных схемах повторяется выбор элемента, имеющего наибольшее (или наименьшее) значение некоторого проверяемого критерия. Параметры выбранного элемента изменяются, и он снова вставляется в список с новым значением критерия, соответствующим новым значениям параметров. Приоритетные очереди часто используются в операционных системах при планировании заданий. Другие типичные применения приоритетных очередей упоминаются в упр. 15, 29 и 36; кроме того, много примеров будет приведено в следующих главах.

Как же реализовать приоритетную очередь? Один из очевидных способов сделать это — организовать и поддерживать порядок по ключам в рассортированном списке элементов. Тогда включение нового элемента, по существу, будет сведено к задаче, рассмотренной при изучении сортировки методом вставок в разделе 5.2.1. При другом, еще более очевидном, способе работы с приоритетной очередью элементы в списке хранятся в произвольном порядке. Тогда для выбора нужного элемента приходится осуществлять поиск наибольшего (или наименьшего) ключа каждый раз, когда необходимо сделать исключение. В обоих этих очевидных подходах неприятность состоит в том, что необходимо порядка $\Omega(N)$ шагов для выполнения либо операции вставки, либо операции удаления, если в списке содержится N элементов, т. е. при больших N эти операции отнимают слишком много времени.

В своей статье о пирамидальной сортировке Уильямс (Williams) обратил внимание на то, что пирамиды идеально подходят для приложений с большими приоритетными очередями, так как элемент можно вставить в пирамиду или удалить из нее за $O(\log N)$ шагов. К тому же все элементы пирамиды компактно располагаются

в последовательных ячейках памяти. Фаза выбора в алгоритме H — это последовательность шагов удаления в процессе наподобие *наибольший из включенных первым исключается*: чтобы исключить наибольший элемент K_1, удаляем его и "протаскиваем" элемент K_N в новой пирамиде из $N-1$ элементов. (Если нужен алгоритм "наименьший из включенных первым исключается", как в примере с лифтом, то, очевидно, можно изменить определение пирамиды, заменив в (3) знак "$\geq$" знаком "$\leq$". Для удобства будем рассматривать здесь лишь случай "наибольший из включенных первым исключается".) Вообще, если требуется исключить наибольший элемент, а затем вставить новый элемент x, можно выполнить процедуру "протаскивания" при $l=1$, $r=N$ и $K=x$. Если же необходимо вставить элемент без предварительного исключения, можно воспользоваться "восходящей" процедурой из упр. 16.

Связанное представление приоритетных очередей. Эффективный способ представления приоритетных очередей в виде связанных бинарных деревьев предложил в 1971 году Кларк Э. Крэйн [см. Clark A. Crane, Technical Report STAN-CS-72-259 (Computer Science Department, Stanford University, 1972)]). Его метод требует наличия в каждой записи двух полей связи и короткого поля счетчика, но по сравнению с пирамидами он обладает следующими преимуществами.

i) Если с приоритетной очередью работают, как со стеком, то операции включения и исключения более эффективны (они отнимают фиксированное время, не зависящее от длины очереди).

ii) Записи никогда не перемещаются, изменяются только указатели.

iii) Можно слить две непересекающиеся приоритетные очереди, содержащие в общей сложности N элементов, всего за $O(\log N)$ шагов.

Немного видоизмененный метод Крэйна проиллюстрирован на рис. 27, на котором показан особый тип структуры бинарного дерева. В каждом узле содержатся поле KEY, поле DIST и два поля связи — LEFT и RIGHT. Поле DIST всегда устанавливается равным длине кратчайшего пути от этого узла до конечного узла (т. е. до пустого узла Λ дерева). Если считать, что DIST(Λ) $=0$ и KEY(Λ) $=-\infty$, то поля KEY и DIST в этом дереве удовлетворяют следующим соотношениям:

$$\mathtt{KEY(P)} \geq \mathtt{KEY(LEFT(P))}, \qquad \mathtt{KEY(P)} \geq \mathtt{KEY(RIGHT(P))}; \tag{9}$$

$$\mathtt{DIST(P)} = 1 + \min(\mathtt{DIST(LEFT(P))}, \mathtt{DIST(RIGHT(P))}); \tag{10}$$

$$\mathtt{DIST(LEFT(P))} \geq \mathtt{DIST(RIGHT(P))}. \tag{11}$$

Соотношение (9) аналогично условию существования пирамиды (3) и служит гарантией того, что в корне дерева находится наибольший ключ, а соотношение (10) — это просто определение поля DIST, сформулированное выше. Соотношение (11) представляет собой интересное новшество: из него следует, что кратчайший путь к конечному узлу всегда можно получить, двигаясь вправо. Будем называть бинарное дерево с этим свойством *левосторонним деревом*, поскольку оно, как правило, сильно "тянется" влево.

Из этих определений ясно, что равенство DIST(P) $=n$ подразумевает существование, по крайней мере, 2^n пустых поддеревьев, расположенных ниже P; в противном случае нашелся бы более короткий путь от P до концевого узла Λ. Таким образом, если в левостороннем дереве имеется N узлов, то путь, ведущий из корня вниз по направлению вправо, содержит не более чем $\lfloor \lg(N+1) \rfloor$ узлов. Новый

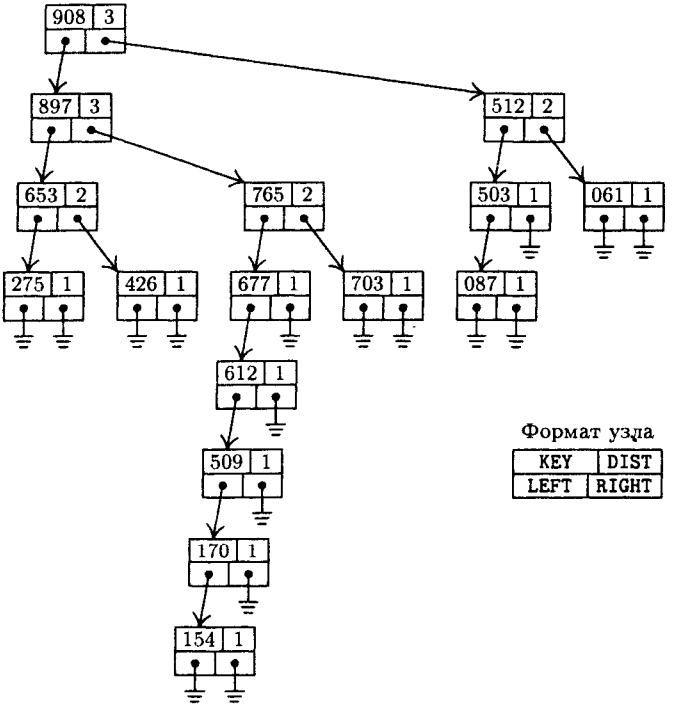


Рис. 27. Приоритетная очередь, представленная в виде левостороннего дерева.

узел можно вставить в приоритетную очередь, пройдя по этому пути (см. упр. 33). Следовательно, в худшем случае необходимо всего $O(\log N)$ шагов. Наилучший случай достигается, когда дерево линейно (все связи RIGHT равны Λ), а наихудший случай — когда оно абсолютно сбалансировано.

Чтобы удалить узел из корня, нужно просто слить два его поддерева. Операция слияния двух непересекающихся левосторонних деревьев, на которые ссылаются указатели P и Q, по своей идее проста: если KEY(P) $\geq$ KEY(Q), то берем в качестве корня P и сливаем Q с правым поддеревом P. При этом изменится DIST(P), а LEFT(P) поменяется местами с RIGHT(P) в случае необходимости. Нетрудно составить подробное описание этого процесса (см. упр. 33).

Сравнение методов работы с приоритетными очередями. Если число узлов N мало́, то для поддержания приоритетной очереди лучше всего прибегнуть к одному из простых методов с использованием линейных списков. Если же N велико, то, очевидно, гораздо более быстрым будет метод, использующий пирамиду или левостороннее дерево, время работы которого составляет порядка $\log N$. В разделе 6.2.3 будет рассмотрен еще один способ представления линейных списков — в виде *сбалансированных деревьев*. Он приводит нас к третьему методу, пригодному для представления приоритетных очередей со временем работы порядка $\log N$. Поэтому уместно сравнить три данных метода.

Мы видели, что операции над левосторонними деревьями, в целом, выполняются несколько быстрее, чем операции над пирамидами, но пирамиды занимают

меньше памяти. Сбалансированные деревья занимают примерно столько же памяти, сколько левосторонние (быть может, чуть меньше); операции над ними выполняются медленнее, чем над пирамидами, и программирование сложнее, но структура сбалансированных деревьев в некоторых отношениях существенно более гибкая. Работая с пирамидами, не так просто предсказать, что произойдет с элементами, если у них равные ключи. Нельзя гарантировать, что элементы с равными ключами будут обрабатываться по принципу "последним пришел — первым вышел" или "первым пришел — первым вышел", если только ключ не расширен и не содержит дополнительного поля "порядковый номер вставки"; тогда равных ключей просто нет. С другой стороны, если применять сбалансированные деревья, можно легко оговорить жесткие условия, относящиеся к равным ключам. Можно также выполнять такие действия, как "вставить x непосредственно перед (или после) y". Сбалансированные деревья симметричны, так что в любой момент можно исключить либо наибольший, либо наименьший элемент, в то время как левосторонние деревья и пирамиды должны быть так или иначе ориентированы. (См., тем не менее, упр. 31, в котором показано, как строить *симметричные* пирамиды.) Сбалансированные деревья можно использовать и для поиска, и для сортировки. Из сбалансированного дерева можно довольно быстро удалять последовательные блоки элементов. Но два сбалансированных дерева нельзя слить менее чем за $\Omega(N)$ шагов, в то время как два левосторонних дерева можно слить всего за $O(\log N)$ шагов.

Итак, сортировка с использованием пирамид наиболее экономна с точки зрения требований к объему памяти; левосторонние деревья хороши тем, что позволяют быстро слить две непересекающиеся приоритетные очереди; и, если нужно, по весьма умеренной цене можно получить гибкость, предоставляемую сбалансированными деревьями.

Со времени выхода в свет "пионерской" работы Уильямса и Крэйна разработано много новых способов представления приоритетных очередей. Теперь в распоряжении программистов имеется большое разнообразие опций, помимо простых списков, левосторонних и сбалансированных деревьев. Некоторые из них перечислены ниже:

- спрямленные деревья, которые обеспечивают обработку симметричной приоритетной очереди всего за $O(\log\log M)$ шагов, если все ключи лежат в заданном диапазоне $0 \le K < M$ [P. van Emde Boas, R. Kaas, E. Zijlstra, *Math. Systems Theory* **10** (1977), 99–127];
- биномиальные очереди [J. Vuillemin, *CACM* **12** (1978), 309–315; M. R. Brown, *SICOMP* **7** (1978), 298–319];
- пагоды [J. Françon, G. Viennot, J. Vuillemin, *FOCS* **19** (1978), 1–7];
- парные пирамиды [M. L. Fredman, R. Sedgewick, D. D. Sleator, R. E. Tarjan, *Algorithmica* **1** (1986), 111–129; J. T. Stasko, J. S. Vitter, *CACM* **30** (1987), 234–249];
- спиральные пирамиды [D. D. Sleator, R. E. Tarjan, *SICOMP* **15** (1986), 52–59];
- пирамиды Фибоначчи [M. L. Fredman, R. E. Tarjan, *JACM* **34** (1987), 596–615] и более общие AF-пирамиды [M. L. Fredman, D. E. Willard, *J. Computer and System Sci.* **48** (1994), 533–551];
- календарные очереди [R. Brown, *CACM* **31** (1988), 1220–1227; G. A. Davison, *CACM* **32** (1989), 1241–1243];

- ослабленные пирамиды [J. R. Driscoll, H. N. Gabow, R. Shrairman, R. E. Tarjan, *CACM* **31** (1988), 1343–1354];
- острóга [M. J. Fischer, M. S. Paterson, *JACM* **41** (1994), 3–30];
- "горячие" очереди [B. V. Cherkassky, A. V. Goldberg, C. Silverstein, *SODA* **8** (1997), 83–92].

Не все из этих методов выдержали испытание временем. Левосторонние деревья сегодня — уже анахронизм; они используются разве что в тех приложениях, в которых необходимо жестко соблюдать дисциплину "последним пришел — первым вышел". Детали реализации биномиальных очередей и пирамид Фибоначчи можно найти в работе D. E. Knuth, The Stanford GraphBase (New York: ACM Press, 1994), 475–489.

***Анализ пирамидальной сортировки.** Алгоритм H до сих пор не был полностью проанализирован математическими методами, но некоторые его свойства можно вывести без особого труда. Поэтому данный раздел завершается довольно подробным анализом пирамид.

На рис. 28 показана форма пирамиды из 26 элементов; каждый узел помечен двоичным числом, соответствующим его индексу в пирамиде. Звездочками на этой диаграмме помечены так называемые *особые узлы*, которые лежат на пути от 1 к N.

Одна из наиболее важных характеристик пирамиды — множество размеров ее поддеревьев. Например, на рис. 28 размеры поддеревьев с корнями в узлах $1, 2, \ldots, 26$ равны соответственно

$$26^*, 15, 10^*, 7, 7, 6^*, 3, 3, 3, 3, 3, 3, 2^*, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1^*. \tag{12}$$

Звездочками помечены *особые поддеревья* с корнями в особых узлах; в упр. 20 показано, что если N имеет двоичное представление

$$N = (b_n b_{n-1} \ldots b_1 b_0)_2, \qquad n = \lfloor \lg N \rfloor, \tag{13}$$

то размеры особых поддеревьев равны

$$(1b_{n-1} \ldots b_1 b_0)_2,\ (1b_{n-2} \ldots b_1 b_0)_2, \ldots, (1b_1 b_0)_2,\ (1b_0)_2,\ (1)_2. \tag{14}$$

Неособые поддеревья всегда абсолютно сбалансированы, так что их размеры имеют вид $2^k - 1$. В упр. 21 показано, что среди неособых поддеревьев существует ровно

$$\left\lfloor \frac{N-1}{2} \right\rfloor \text{ размером } 1,\ \left\lfloor \frac{N-2}{4} \right\rfloor \text{ размером } 3,\ \left\lfloor \frac{N-4}{8} \right\rfloor \text{ размером } 7,\ \ldots,$$
$$\left\lfloor \frac{N-2^{n-1}}{2^n} \right\rfloor (2^n - 1) \text{ размера.} \tag{15}$$

Например, на рис. 28 изображено двенадцать неособых поддеревьев размером 1, шесть поддеревьев размером 3, два — размером 7 и одно — размером 15.

Пусть s_l — размер поддерева с корнем l, а M_N — мультимножество $\{s_1, s_2, \ldots, s_N\}$ всех этих размеров. Используя (14) и (15), легко вычислить M_N при любом заданном N. В упр. 5.1.4–20 показано, что общее число способов построения пирамиды из целых чисел $\{1, 2, \ldots, N\}$ равно

$$N!/s_1 s_2 \ldots s_N = N! \Big/ \prod \{s \mid s \in M_N\}. \tag{16}$$

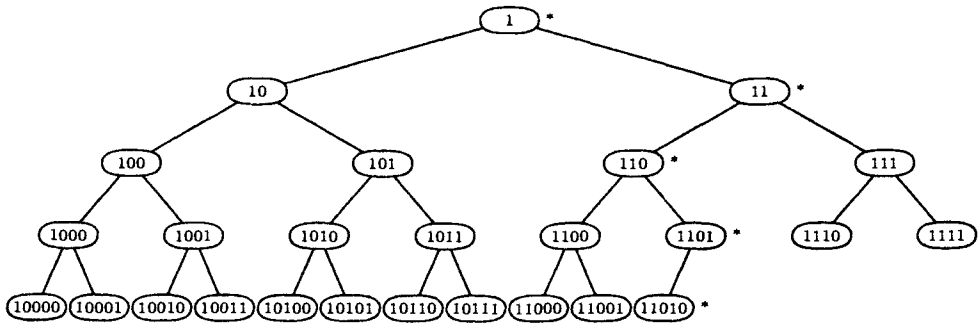


Рис. 28. Так выглядит пирамида из $26 = (11010)_2$ элементов.

Например, количество способов такого расположения 26 букв $\{A, B, C, \ldots, Z\}$ на рис. 28, чтобы по вертикали сохранялся алфавитный порядок, составляет

$$26!/(26 \cdot 10 \cdot 6 \cdot 2 \cdot 1 \cdot 1^{12} \cdot 3^6 \cdot 7^2 \cdot 15^1).$$

Теперь можно проанализировать фазу построения пирамиды в алгоритме H, т. е. вычислительные операции, которые завершаются до того, как на шаге H2 впервые выполнится условие $l = 1$. К счастью, благодаря сформулированной ниже теореме анализ построения пирамиды можно свести к изучению независимых операций "протаскивания".

Теорема H. *Если исходными данными для алгоритма H служит случайная перестановка множества $\{1, 2, \ldots, N\}$, то в фазе построения пирамиды с одинаковой вероятностью может получиться любая из $N!/\prod\{s \mid s \in M_N\}$ возможных пирамид. Более того, все $\lfloor N/2 \rfloor$ операций протаскивания, выполненных в течение этой фазы, равномерны в том смысле, что по достижении шага H8 все s_l возможных значений переменной i равновероятны.*

Доказательство. Применим метод, который в численном анализе называется методом обратной задачи. Пусть в качестве одного из возможных результатов выполнения протаскивания задана пирамида $K_1 \ldots K_N$ с корнем в узле l. Тогда ясно, что имеется точно s_l исходных конфигураций $K'_1 \ldots K'_N$ массива, которые после протаскивания дают такой результат. Все эти исходные конфигурации имеют различные значения K'_l; следовательно, рассуждая наоборот, можно получить ровно $s_l\, s_{l+1} \ldots s_N$ исходных перестановок множества $\{1, 2, \ldots, N\}$, которые после завершения протаскивания в позицию l дают конфигурацию $K_1 \ldots K_N$.

Случай, когда $l = 1$, типичен: пусть $K_1 \ldots K_N$ — пирамида и пусть $K'_1 \ldots K'_N$ — массив, который преобразуется в $K_1 \ldots K_N$ в результате протаскивания при $l = 1$, $K = K'_1$. Если $K = K_i$, то должны иметь место равенства $K'_i = K_{\lfloor i/2 \rfloor}$, $K'_{\lfloor i/2 \rfloor} = K_{\lfloor i/4 \rfloor}$ и т. д. При этом $K'_j = K_j$ для всех j, не лежащих на пути от 1 к i. Обратно, при любом i в результате подобного построения получается массив $K'_1 \ldots K'_N$, такой, что (a) операция протаскивания приводит к преобразованию массива $K'_1 \ldots K'_N$ в $K_1 \ldots K_N$, и (b) $K_{\lfloor j/2 \rfloor} \geq K_j$ при $2 \leq \lfloor j/2 \rfloor < j \leq N$. Следовательно, возможно ровно N таких массивов $K'_1 \ldots K'_N$ и операция протаскивания имеет равномерное распределение. (Пример доказательства этой теоремы приводится в упр. 22.) ∎

Обратившись к параметрам A, B, C, D в анализе программы H, можно видеть, что равномерная операция протаскивания, выполняемая по отношению к поддереву размером s, дает вклад $\lfloor s/2 \rfloor / s$ в среднее значение величины A. Ее вклад в среднее значение величины B равен

$$\frac{1}{s}(0+1+1+2+\cdots+\lfloor \lg s \rfloor) = \frac{1}{s}\sum_{k=1}^{s}\lfloor \lg k \rfloor = \frac{1}{s}\big((s+1)\lfloor \lg s \rfloor - 2^{\lfloor \lg s \rfloor + 1} + 2\big)$$

(см. упр. 1.2.4–42), и она дает вклад либо $2/s$, либо 0 в среднее значение параметра D в зависимости от того, каким является s: четным или нечетным. Несколько сложнее определить соответствующий вклад в среднее значение параметра C, так что эту задачу мы предлагаем читателю решить самостоятельно (см. упр. 26). Производя суммирование по всем операциям протаскивания, находим, что среднее значение параметра A за время построения пирамиды равно

$$A'_N = \sum\{\lfloor s/2 \rfloor / s \mid s \in M_N\}. \tag{17}$$

Аналогичные формулы имеют место и для B, C и D, так что можно без особого труда точно вычислить эти средние значения. В следующей таблице приведены типичные результаты.

N	A'_N	B'_N	C'_N	D'_N
99	19.18	68.35	42.95	0.00
100	19.93	69.39	42.71	1.84
999	196.16	734.66	464.53	0.00
1000	196.94	735.80	464.16	1.92
9999	1966.02	7428.18	4695.54	0.00
10000	1966.82	7429.39	4695.06	1.97
10001	1966.45	7430.07	4695.84	0.00
10002	1967.15	7430.97	4695.95	1.73

Что касается асимптотики, то в M_N можно не обращать внимания на размеры особых поддеревьев, и тогда мы найдем, например, что

$$A'_N = \frac{N}{2}\cdot\frac{0}{1} + \frac{N}{4}\cdot\frac{1}{3} + \frac{N}{8}\cdot\frac{3}{7} + \cdots + O(\log N) = \big(1 - \tfrac{1}{2}\alpha\big)N + O(\log N), \tag{18}$$

где

$$\alpha = \sum_{k\ge 1}\frac{1}{2^k - 1} = 1.60669\,51524\,15291\,76378\,33015\,23190\,92458\,04805\text{–}. \tag{19}$$

(Это значение получил Дж. У. Ренч (мл.) (J. W. Wrench, Jr.), пользуясь преобразованием ряда из упр. 27. Пол Эрдеш (Paul Erdős) доказал, что α является иррациональным числом [*J. Indian Math. Soc.* **12** (1948), 63–66], а Питер Борвейн (Peter Borwein) продемонстрировал иррациональный характер многих других констант [*Proc. Camb. Phil. Soc.* **112** (1992), 141–146].) При больших N можно использовать приближенные формулы

$$
\begin{aligned}
A'_N &\approx 0.1967N + (-1)^N 0.3;\\
B'_N &\approx 0.74403N - 1.3\ln N;\\
C'_N &\approx 0.47034N - 0.8\ln N;\\
D'_N &\approx (1.8 \pm 0.2)[N \text{ четное}].
\end{aligned}
\tag{20}
$$

Нетрудно определить также максимальные и минимальные значения. Для построения пирамиды требуется всего $O(N)$ шагов (см. упр. 23).

Этим, по существу, завершается анализ фазы построения пирамиды в алгоритме H. Анализ фазы выбора — совсем другая задача, которая еще ожидает своего решения! Пусть пирамидальная сортировка применяется к N элементам; обозначим через A''_N, B''_N, C''_N и D''_N средние значения величин A, B, C и D во время фазы выбора. Поведение алгоритма H подвержено сравнительно малым колебаниям около эмпирически установленных средних значений

$$
\begin{aligned}
A''_N &\approx 0.152N;\\
B''_N &\approx N\lg N - 2.61N;\\
C''_N &\approx \tfrac{1}{2}N\lg N - 1.41N;\\
D''_N &\approx \lg N \pm 2.
\end{aligned}
\tag{21}
$$

Тем не менее до сих пор не найдено адекватного теоретического объяснения поведения D''_N или эмпирически подобранных значений констант 0.152, 2.61 и 1.41. Ведущие члены в выражениях для B''_N и C''_N, однако, очень изящно обоснованы Р. Шаффером и Р. Седгевиком (см. упр. 30). Шаффер также доказал, что минимальное и максимальное значения C''_N равны $\frac{1}{4}N\lg N$ и $\frac{3}{4}N\lg N$ соответственно.

УПРАЖНЕНИЯ

1. [*10*] Является ли сортировка посредством простого выбора (алгоритм S) устойчивой?

2. [*15*] Почему в алгоритме S более удобно находить сначала наибольший элемент, затем — наибольший из оставшихся и т. д., вместо того чтобы находить сначала наименьший элемент, затем — наименьший из оставшихся и т. д.?

3. [*M21*] (a) Докажите, что если алгоритм S применяется к случайной перестановке множества $\{1, 2, \ldots, N\}$, то в результате первого выполнения шагов S2 и S3 получается случайная перестановка множества $\{1, 2, \ldots, N-1\}$, за которой следует N. (Иначе говоря, массив $K_1 \ldots K_{N-1}$ с одинаковой вероятностью может быть любой перестановкой множества $\{1, 2, \ldots, N-1\}$.) (b) Следовательно, если через B_N обозначить среднее значение величины B в программе S, то при условии, что исходный массив упорядочен случайным образом, имеем $B_N = H_N - 1 + B_{N-1}$. [*Указание.* См. выражение 1.2.10–(16).]

▸ **4.** [*M25*] На шаге S3 алгоритма S ничего не происходит, если $i = j$. Не лучше ли перед выполнением шага S3 проверить условие $i = j$? Чему равно среднее число случаев выполнения условия $i = j$ на шаге S3, если исходный массив случаен?

5. [*20*] Чему равно значение параметра B в анализе программы S для исходного массива $N \ldots 3\,2\,1$?

6. [*M29*] (a) Пусть $a_1\, a_2 \ldots a_N$ — перестановка множества $\{1, 2, \ldots, N\}$ с C циклами, I инверсиями и такая, что при ее сортировке с помощью программы S выполняется B обменов на правосторонний максимум. Докажите, что $2B \le I + N - C$. [*Указание.* См. упр. 5.2.2–1.] (b) Покажите, что $I + N - C \le \lfloor N^2/2 \rfloor$; следовательно, B не превышает $\lfloor N^2/4 \rfloor$.

7. [*M41*] Найдите дисперсию параметра B в программе S как функцию от N, считая, что исходный массив случаен.

▶ **8.** [*24*] Покажите, что если при поиске $\max(K_1, \ldots, K_j)$ на шаге S2 просматривать ключи слева направо $(K_1, K_2, \ldots, K_j)$, а не наоборот, как в программе S, то можно сократить число сравнений при следующих итерациях шага S2. Напишите MIX-программу, реализующую этот подход.

9. [*M25*] Чему равно среднее число сравнений, выполняемых алгоритмом из упр. 8 для случайного исходного массива?

10. [*12*] Как будет выглядеть дерево, изображенное на рис. 23, после вывода 14 из 16 первоначальных элементов?

11. [*10*] Как будет выглядеть дерево, изображенное на рис. 24, после вывода элемента 908?

12. [*M20*] Сколько раз будет выполнено сравнение $-\infty$ с $-\infty$, если применить восходящий метод, представленный на рис. 23, для упорядочения массива из 2^n элементов?

13. [*20*] (Дж. У. Дж. Уильямс (J. W. J. Williams).) На шаге H4 алгоритма H различаются три случая: $j < r$, $j = r$ и $j > r$. Покажите, что если $K \geq K_{r+1}$, то можно так упростить шаг H4, что разветвление будет происходить лишь по двум ветвям. Как следует изменить шаг H2, чтобы обеспечить в процессе пирамидальной сортировки выполнение условия $K \geq K_{r+1}$?

14. [*10*] Покажите, что простая очередь — частный случай приоритетной. (Объясните, какие ключи нужно присваивать элементам, чтобы процедура "наибольший из включенных первым исключается" была эквивалентна процедуре "первым пришел — первым вышел"?) Является ли стек также частным случаем приоритетной очереди?

▶ **15.** [*M22*] (Б. Э. Чартрс (B. A. Chartres).) Придумайте быстрый алгоритм построения таблицы простых чисел $\leq N$, в котором используется приоритетная очередь, чтобы избежать операций деления. [*Указание.* Пусть наименьший ключ в приоритетной очереди будет наименьшим нечетным непростым числом, бóльшим, чем самое последнее нечетное число, рассматриваемое как кандидат в простые числа. Попытайтесь свести к минимуму количество элементов в этой очереди.]

16. [*20*] Постройте эффективный алгоритм, который позволяет вставить новый ключ в данную пирамиду из n элементов, порождая пирамиду из $n+1$ элементов.

17. [*20*] Алгоритм из упр. 16 можно использовать для построения пирамиды вместо метода "уменьшать l до 1", применяемого в алгоритме H. Порождают ли оба метода из одного и того же исходного массива одну и ту же пирамиду?

▶ **18.** [*21*] (Р. У. Флойд (R. W. Floyd).) Во время фазы выбора в алгоритме пирамидальной сортировки ключ K, как правило, принимает довольно малые значения, и поэтому почти при всех сравнениях на шаге H6 обнаруживается, что $K < K_j$. Как можно изменить алгоритм, чтобы ключ K не сравнивался с K_j в основном цикле вычислений, и таким образом почти наполовину уменьшить число сравнений?

19. [*21*] Предложите алгоритм *исключения* данного элемента из пирамиды размером N, порождающий пирамиду размером $N-1$.

20. [*M20*] Покажите, что формулы (14) задают размеры особых поддеревьев пирамиды.

21. [*M24*] Докажите, что формулы (15) задают размеры неособых поддеревьев пирамиды.

▶ **22.** [*20*] Какие перестановки множества $\{1, 2, 3, 4, 5\}$ на фазе построения пирамиды в алгоритме H преобразуются в 5 3 4 1 2?

23. [*M28*] (a) Докажите, что длина пути B в алгоритме протаскивания никогда не превышает $\lfloor \lg(r/l) \rfloor$. (b) Согласно неравенствам (8) ни при каком конкретном применении

алгоритма H величина B не может превзойти $N\lfloor \lg N \rfloor$. Найдите максимальное значение B по всевозможным исходным массивам как функцию от N. (Нужно доказать, что существует такой исходный массив, на котором B принимает это максимальное значение.)

24. [*M24*] Выведите точную формулу стандартного отклонения величины B'_N (суммарная длина пути, пройденного по дереву во время фазы построения пирамиды в алгоритме H).

25. [*M20*] Чему равен средний вклад в значение параметра C за время первой операции протаскивания, когда $l = 1$ и $r = N$, если $N = 2^{n+1} - 1$?

26. [*M30*] Выполните упр. 25: (a) для $N = 26$, (b) для произвольного N.

27. [*M25*] (Т. Клаузен (T. Clausen), 1828.) Докажите, что

$$\sum_{n\geq 1} \frac{x^n}{1-x^n} = \sum_{n\geq 1} \frac{1+x^n}{1-x^n} x^{n^2}.$$

(Положив $x = \frac{1}{2}$, получите очень быстро сходящийся ряд для вычисления (19).)

28. [*35*] Продумайте идею *тернарных пирамид*, основанных на полных тернарных, а не бинарных деревьях. Будет ли тернарная пирамидальная сортировка выполняться быстрее бинарной?

29. [*26*] (У. С. Браун (W. S. Brown).) Постройте алгоритм умножения многочленов или степенных рядов $(a_1x^{i_1} + a_2x^{i_2} + \cdots)(b_1x^{j_1} + b_2x^{j_2} + \cdots)$, который порождал бы коэффициенты произведения $c_1x^{i_1+j_1} + \cdots$ в том порядке, в котором перемножаются коэффициенты исходных многочленов. [*Указание.* Воспользуйтесь подходящей приоритетной очередью.]

▸ **30.** [*HM35*] (Р. Шаффер (R. Schaffer) и Р. Седгевик (R. Sedgewick).) Пусть h_{nm} — число пирамид элементов $\{1, 2, \ldots, n\}$, для которых фаза выбора даст ровно m продвижений. Докажите, что

$$h_{nm} \leq 2^m \prod_{k=2}^{n} \lg k,$$

и используйте это соотношение для того, чтобы показать, что среднее число продвижений, выполняемых алгоритмом H, равно $N \lg N + O(N \log\log N)$.

31. [*37*] (Дж. У. Дж. Уильямс.) Покажите, что если две пирамиды подходящим образом совместить "основание к основанию", то появится возможность поддерживать структуру, в которой в любой момент можно за $O(\log n)$ шагов исключить либо наибольший, либо наименьший элемент. (Такую структуру можно назвать *приоритетным деком*.)

32. [*M28*] Докажите, что число продвижений B при пирамидальной сортировке всегда оказывается не меньше $\frac{1}{2}N \lg N + O(N)$, если все сортируемые ключи различны. *Указание.* Рассмотрите продвижение $\lceil N/2 \rceil$ наибольших ключей.

33. [*21*] Разработайте алгоритм слияния двух непересекающихся приоритетных очередей, представленных в виде левосторонних деревьев, в одну. (В частности, если одна из данных очередей содержит всего один элемент, то ваш алгоритм будет вставлять его в другую очередь.)

34. [*M41*] Сколько можно построить левосторонних деревьев из N узлов, если игнорировать значения поля KEY? Эта последовательность начинается с чисел 1, 1, 2, 4, 8, 17, 38, 87, 203, 482, 1160, ...; покажите, что данное число асимптотически стремится к $ab^N N^{-3/2}$ при соответствующим образом выбранных константах a и b, используя методику, аналогичную примененной в упр. 2.3.4.4–4.

35. [*26*] Если в левостороннее дерево добавить связи UP (ср. с анализом деревьев с тремя связями в разделе 6.2.3), то получим возможность исключать из приоритетной очереди произвольный узел P следующим образом: слить LEFT(P) и RIGHT(P) и поместить полученное поддерево на место P, затем исправлять поля DIST предков узла P до тех пор, пока не будет достигнут либо корень, либо узел, поле DIST которого не меняется.

Докажите, что при этом никогда не потребуется изменить более чем $O(\log N)$ полей DIST, несмотря даже на то, что дерево может содержать очень длинные восходящие пути.

36. [*18*] (*Замещение наиболее давно использованной страницы.*) Во многих операционных системах используется алгоритм следующего типа: над набором узлов допустимы две операции: (i) использование узла и (ii) замещение наиболее давно использованного узла новым. Какая структура данных облегчает нахождение наиболее давно использованного узла?

37. [*HM32*] Пусть $e_N(k)$ — предполагаемое расстояние между самым большим k-м элементом и корнем в случайной пирамиде из N элементов и пусть $e(k) = \lim_{N\to\infty} e_N(k)$. Тогда $e(1) = 0$, $e(2) = 1$, $e(3) = 1.5$, а $e(4) = 1.875$. Найдите значение асимптотического выражения для $e(k)$ через $O(k^{-1})$.

38. [*M21*] Найдите простое рекуррентное соотношение для мультимножества M_N размеров поддеревьев в пирамиде или в полном бинарном дереве, имеющем N внутренних узлов.

5.2.4. Сортировка методом слияния

*Слияние** означает объединение двух или более упорядоченных массивов в один упорядоченный массив. Можно, например, слить подмассивы 503 703 765 и 087 512 677 и получить 087 503 512 677 703 765. Простой способ сделать это — сравнить два наименьших элемента, вывести наименьший из них и повторить эту процедуру. Начав с

$$\begin{cases} 503\ 703\ 765 \\ 087\ 512\ 677 \end{cases},$$

получим

$$087\ \begin{cases} 503\ 703\ 765 \\ 512\ 677 \end{cases},$$

затем

$$087\ 503\ \begin{cases} 703\ 765 \\ 512\ 677 \end{cases}$$

и

$$087\ 503\ 512\ \begin{cases} 703\ 765 \\ 677 \end{cases}$$

и т. д. Необходимо решить, что делать в случае, когда исчерпается один из массивов. Весь процесс подробно описан в следующем алгоритме.

Алгоритм M (*Двухпутевое слияние*). Этот алгоритм осуществляет слияние упорядоченных массивов $x_1 \le x_2 \le \cdots \le x_m$ и $y_1 \le y_2 \le \cdots \le y_n$ в массив $z_1 \le z_2 \le \cdots \le z_{m+n}$ (рис. 29).

M1. [Начальная установка.] Установить $i \leftarrow 1$, $j \leftarrow 1$, $k \leftarrow 1$.

---

* В англоязычной литературе термину *слияние* соответствуют два равнозначных термина — *merging* и *collating*. — *Прим. перев.*

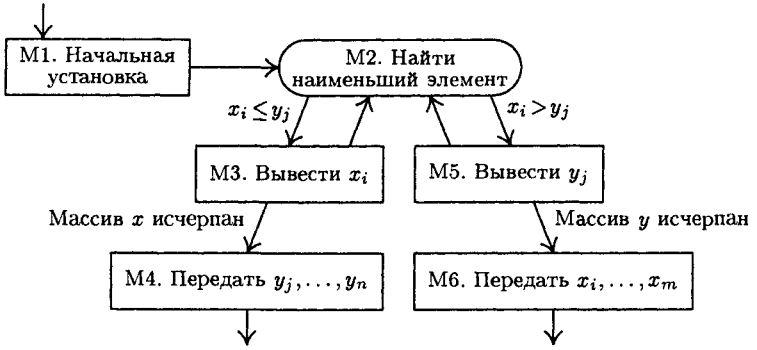


Рис. 29. Слияние $x_1 \le \cdots \le x_m$ с $y_1 \le \cdots \le y_n$.

M2. [Найти наименьший элемент.] Если $x_i \le y_j$, перейти к шагу M3; в противном случае перейти к шагу M5.

M3. [Вывести x_i.] Установить $z_k \leftarrow x_i$, $k \leftarrow k+1$, $i \leftarrow i+1$. Если $i \le m$, вернуться к шагу M2.

M4. [Передать $y_j, \ldots, y_n$.] Установить $(z_k, \ldots, z_{m+n}) \leftarrow (y_j, \ldots, y_n)$ и завершить выполнение алгоритма.

M5. [Вывести y_j.] Установить $z_k \leftarrow y_j$, $k \leftarrow k+1$, $j \leftarrow j+1$. Если $j \le n$, вернуться к шагу M2.

M6. [Передать $x_i, \ldots, x_m$.] Установить $(z_k, \ldots, z_{m+n}) \leftarrow (x_i, \ldots, x_m)$ и завершить выполнение алгоритма. ▌

В разделе 5.3.2 будет показано, что эта простая процедура, по существу, — наилучший из возможных способов слияния на компьютере с обычной архитектурой, когда $m \approx n$. (Но, если m гораздо меньше n, можно разработать более эффективные алгоритмы сортировки, хотя в общем случае они довольно сложны.) Алгоритм M без существенной потери эффективности можно немного упростить, добавив в конец исходных массивов искусственных "стражей" (ограничивающие элементы $x_{m+1} = y_{n+1} = \infty$) и остановившись перед выводом ∞. Анализ алгоритма M приведен в упр. 2.

Общий объем операций, выполняемых алгоритмом M, по существу, пропорционален $m+n$, откуда понятно, почему считается, что слияние — более простая задача, чем сортировка. Однако задачу сортировки можно свести к слияниям, сливая все более длинные подмассивы до тех пор, пока не будет рассортирован весь массив. Такой подход можно рассматривать как развитие идеи сортировки методом вставок: вставка нового элемента в упорядоченный массив — частный случай слияния при $n = 1$. Чтобы ускорить процесс вставок, можно рассмотреть вставку нескольких элементов за один раз, группируя несколько операций, а это естественным образом приведет к общей идее сортировки методом слияния. С исторической точки зрения метод слияний — один из самых первых методов сортировки при помощи компьютеров; он был предложен Джоном фон Нейманом (John von Neumann) еще в 1945 году (см. раздел 5.5).

Довольно подробно слияние рассматривается в разделе 5.4 в связи с алгоритмами внешней сортировки, а в настоящем разделе речь пойдет о сортировке в оперативной памяти с произвольным доступом.

В табл. 1 проиллюстрирована сортировка методом слияния, когда "свечка сжигается с обоих концов", подобно тем процедурам просмотра элементов массива, которые применялись при быстрой сортировке, поразрядной обменной сортировке и т. д. Будем анализировать исходный массив слева и справа, двигаясь к середине. Пропустим пока первую строку и рассмотрим переход от второй строки к третьей. Слева мы видим восходящую серию 503 703 765, а справа, если читать справа налево, имеем серию 087 512 677. Слияние этих двух последовательностей дает подмассив 087 503 512 677 703 765, который помещается слева в строке 3. Затем ключи 061 612 908 в строке 2 сливаются с 170 509 897 и результат (061 170 509 612 897 908) записывается *справа* в строке 3. Наконец, 154 275 426 653 сливается с 653 (перекрытие обнаруживается прежде, чем оно может привести к нежелательным последствиям) и результат записывается слева. Точно так же строка 2 получается из строки 1 исходного массива.

Таблица 1

СОРТИРОВКА МЕТОДОМ ЕСТЕСТВЕННОГО ДВУХПУТЕВОГО СЛИЯНИЯ

503	087	512	061	908	170	897	275	653	426	154	509	612	677	765	703
503	703	765	061	612	908	154	275	426	653	897	509	170	677	512	087
087	503	512	677	703	765	154	275	426	653	908	897	612	509	170	061
061	087	170	503	509	512	612	677	703	765	897	908	653	426	275	154
061	087	154	170	275	426	503	509	512	612	653	677	703	765	897	908

Вертикальными линиями в табл. 1 отмечены границы между сериями. Это так называемые *ступеньки вниз*, где меньший элемент следует за большим. В общем случае в середине массива возникает двусмысленная ситуация, когда при движении с обоих концов считывается один и тот же ключ; это не приведет к осложнениям, если проявить чуточку осторожности, как в следующем алгоритме. Такой метод по традиции называется "естественным" слиянием, потому что в нем используются серии, которые "естественно" образуются в исходном массиве.

Алгоритм N (*Сортировка методом естественного двухпутевого слияния*). При сортировке записей $R_1, \ldots, R_N$ используются две области памяти, в каждой из которых может содержаться N записей. Для удобства обозначим записи, находящиеся во второй области, через $R_{N+1}, \ldots, R_{2N}$, хотя в действительности запись R_{N+1} может и не примыкать непосредственно к R_N. Начальное содержимое записей $R_{N+1}, \ldots, R_{2N}$ не имеет значения. После завершения сортировки ключи будут упорядочены таким образом: $K_1 \le \cdots \le K_N$ (рис. 30).

N1. [Начальная установка.] Установить $s \leftarrow 0$. (При $s = 0$ будем пересылать записи из области $(R_1, \ldots, R_N)$ в область $(R_{N+1}, \ldots, R_{2N})$; при $s = 1$ области по отношению к пересылкам поменяются ролями.)

N2. [Подготовка к просмотру.] Если $s = 0$, установить $i \leftarrow 1$, $j \leftarrow N$, $k \leftarrow N + 1$, $l \leftarrow 2N$; если $s = 1$, установить $i \leftarrow N + 1$, $j \leftarrow 2N$, $k \leftarrow 1$, $l \leftarrow N$. (Переменные i, j, k, l указывают текущие позиции во "входных массивах", из которых выполняется чтение, и в "выходных массивах", в которые осуществляется запись.) Установить $d \leftarrow 1$, $f \leftarrow 1$. (Переменная d задает текущее направление вывода; f устанавливается равной 0, если необходимы дальнейшие просмотры.)

N3. [Сравнение $K_i : K_j$.] Если $K_i > K_j$, перейти к шагу N8. Если $i = j$, установить $R_k \leftarrow R_i$ и перейти к шагу N13.

N4. [Пересылка R_i.] (Шаги N4–N7 аналогичны шагам M3–M4 алгоритма M.) Установить $R_k \leftarrow R_i$, $k \leftarrow k + d$.

N5. [Ступенька вниз?] Увеличить i на 1. Далее, если $K_{i-1} \leq K_i$, вернуться к шагу N3.

N6. [Пересылка R_j.] Установить $R_k \leftarrow R_j$, $k \leftarrow k + d$.

N7. [Ступенька вниз?] Уменьшить j на 1. Далее, если $K_{j+1} \leq K_j$, вернуться к шагу N6; иначе — перейти к шагу N12.

N8. [Пересылка R_j.] (Шаги N8–N11 двойственны шагам N4–N7.) Установить $R_k \leftarrow R_j$, $k \leftarrow k + d$.

N9. [Ступенька вниз?] Уменьшить j на 1. Далее, если $K_{j+1} \leq K_j$, вернуться к шагу N3.

N10. [Пересылка R_i.] Установить $R_k \leftarrow R_i$, $k \leftarrow k + d$.

N11. [Ступенька вниз?] Увеличить i на 1. Далее, если $K_{i-1} \leq K_i$, вернуться к шагу N10.

N12. [Переключение направления.] Установить $f \leftarrow 0$, $d \leftarrow -d$ и выполнить замену $k \leftrightarrow l$. Вернуться к шагу N3.

N13. [Переключение областей.] Если $f = 0$, установить $s \leftarrow 1 - s$ и вернуться к шагу N2. В противном случае сортировка будет завершена. Если $s = 0$, установить $(R_1, \ldots, R_N) \leftarrow (R_{N+1}, \ldots, R_{2N})$. (Если результат можно оставить в области $(R_{N+1}, \ldots, R_{2N})$, примерно в половине случаев последнее копирование оказывается необязательным.) ▮

В этом алгоритме есть одна небольшая тонкость, которая объясняется в упр. 5.

Запрограммировать алгоритм N для машины MIX нетрудно, но можно проанализировать ход его выполнения и без разработки программы. Если массив случаен, то в нем имеется около $\frac{1}{2}N$ восходящих серий, так как $K_i > K_{i+1}$ с вероятностью $\frac{1}{2}$. Подробная информация о количестве серий при несколько отличных предположениях была получена в разделе 5.1.3. При каждом просмотре (на каждом проходе алгоритма) число серий сокращается вдвое (за исключением необычных случаев, таких, как ситуация, описанная в упр. 6). Следовательно, число просмотров составляет, как правило, около $\lg \frac{1}{2}N = \lg N - 1$. При каждом просмотре необходимо переписать все N записей, и, как показано в упр. 2, бóльшая часть времени затрачивается на выполнение шагов N3–N5, N8, N9. Если считать, что равные ключи встречаются с малой вероятностью, то время, затрачиваемое во внутреннем цикле, можно охарактеризовать следующим образом.

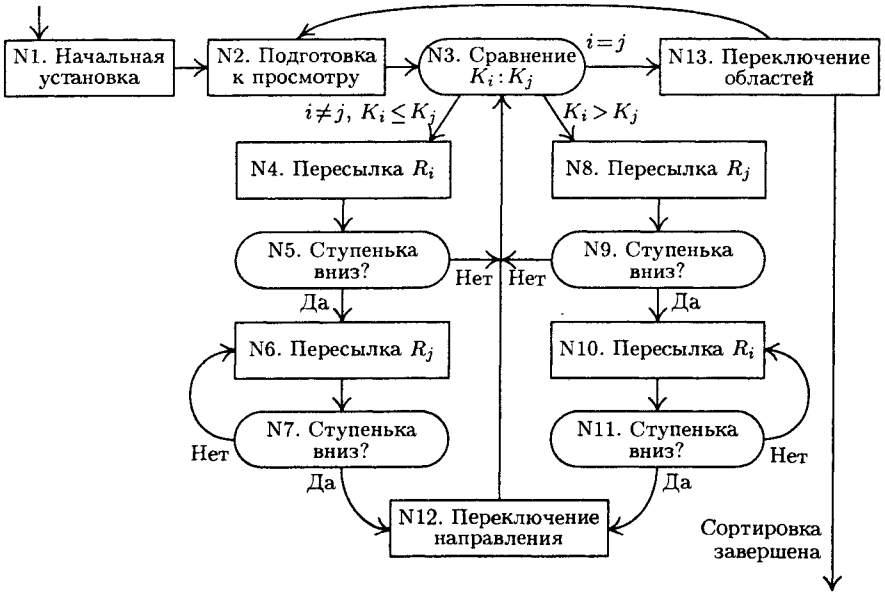


Рис. 30. Сортировка методом слияния.

	Шаг	Операции	Время
	$N3$	CMPA, JG, JE	$3.5u$
Либо	$N4$	STA, INC	$3u$
	$N5$	INC, LDA, CMPA, JGE	$6u$
Либо	$N8$	STX, INC	$3u$
	$N9$	DEC, LDX, CMPX, JGE	$6u$

Значит, при каждом просмотре на каждую запись затрачивается 12.5 машинных циклов и общее время выполнения асимптотически приближается к $12.5N \lg N$ как в среднем, так и в наихудшем случаях. Это происходит медленнее быстрой сортировки и не настолько лучше времени работы пирамидальной сортировки, чтобы оправдать вдвое больший расход памяти, так как асимптотическое время выполнения программы 5.2.3H равно $18N \lg N$.

В алгоритме N границы между сериями полностью определяются ступеньками вниз. Такой подход обладает тем потенциальным преимуществом, что исходные массивы с преобладанием возрастающего или убывающего расположения элементов могут обрабатываться очень быстро, но при этом замедляется основной цикл вычислений. Вместо того чтобы проверять ступеньки вниз, длину серий можно установить принудительно: считать, что все серии исходного массива имеют длину 1, после первого просмотра все серии (кроме, возможно, последней) имеют длину $2, \ldots$, после k-го просмотра все серии (кроме, возможно, последней) имеют длину 2^k. В отличие от "естественного" слияния в алгоритме N такой способ называется *простым* двухпутевым слиянием.

Алгоритм простого двухпутевого слияния очень напоминает алгоритм N — он, по существу, описывается той же блок-схемой. Тем не менее методы отличаются один от другого, и поэтому стоит записать весь алгоритм целиком.

Алгоритм S (*Сортировка методом простого двухпутевого слияния*). Как и в алгоритме N, при сортировке записей $R_1, \ldots, R_N$ используются две области памяти.

S1. [Начальная установка.] Установить $s \leftarrow 0$, $p \leftarrow 1$. (Смысл переменных s, i, j, k, l и d приводится в описании алгоритма N. Здесь p — размер восходящих серий, которые будут сливаться во время текущего просмотра. Другие переменные q и r предназначены для отслеживания количества неслитых элементов в сериях в ходе выполнения алгоритма.)

S2. [Подготовка к просмотру.] Если $s = 0$, установить $i \leftarrow 1$, $j \leftarrow N$, $k \leftarrow N$, $l \leftarrow 2N + 1$; если $s = 1$, установить $i \leftarrow N + 1$, $j \leftarrow 2N$, $k \leftarrow 0$, $l \leftarrow N + 1$. Далее установить $d \leftarrow 1$, $q \leftarrow p$, $r \leftarrow p$.

S3. [Сравнение $K_i : K_j$.] Если $K_i > K_j$, перейти к шагу S8.

S4. [Пересылка R_i.] Установить $k \leftarrow k + d$, $R_k \leftarrow R_i$.

S5. [Конец серии?] Установить $i \leftarrow i + 1$, $q \leftarrow q - 1$. Если $q > 0$, вернуться к шагу S3.

S6. [Пересылка R_j.] Установить $k \leftarrow k + d$. Далее, если $k = l$, перейти к шагу S13; в противном случае установить $R_k \leftarrow R_j$.

S7. [Конец серии?] Установить $j \leftarrow j - 1$, $r \leftarrow r - 1$. Если $r > 0$, вернуться к шагу S6; в противном случае перейти к шагу S12.

S8. [Пересылка R_j.] Установить $k \leftarrow k + d$, $R_k \leftarrow R_j$.

S9. [Конец серии?] Установить $j \leftarrow j - 1$, $r \leftarrow r - 1$. Если $r > 0$, вернуться к шагу S3.

S10. [Пересылка R_i.] Установить $k \leftarrow k + d$. Далее, если $k = l$, перейти к шагу S13; в противном случае установить $R_k \leftarrow R_i$.

S11. [Конец серии?] Установить $i \leftarrow i + 1$, $q \leftarrow q - 1$. Если $q > 0$, вернуться к шагу S10.

S12. [Переключение направления.] Установить $q \leftarrow p$, $r \leftarrow p$, $d \leftarrow -d$ и выполнить замену $k \leftrightarrow l$. Если $j - i < p$, вернуться к шагу S10; в противном случае вернуться к шагу S3.

S13. [Переключение областей.] Установить $p \leftarrow p + p$. Если $p < N$, установить $s \leftarrow 1 - s$ и вернуться к шагу S2. В противном случае сортировка будет завершена. Если $s = 0$, установить

$$(R_1, \ldots, R_N) \leftarrow (R_{N+1}, \ldots, R_{2N}).$$

(Последнее копирование необходимо выполнять тогда и только тогда, когда $\lceil \lg N \rceil$ нечетно, независимо от распределения записей в исходном массиве. Таким образом, можно заранее предсказать положение рассортированного массива и копирование, скорее всего, станет излишним.) ▮

Пример выполнения алгоритма представлен в табл. 2. Довольно удивительно, что этот метод прекрасно работает и тогда, когда N не является степенью 2. Не все

Таблица 2

СОРТИРОВКА МЕТОДОМ ПРОСТОГО ДВУХПУТЕВОГО СЛИЯНИЯ

503	087	512	061	908	170	897	275	653	426	154	509	612	677	765	703
503	703	512	677	509	908	426	897	653	275	170	154	612	061	765	087
087	503	703	765	154	170	509	908	897	653	426	275	677	612	512	061
061	087	503	512	612	677	703	765	908	897	653	509	426	275	170	154
061	087	154	170	275	426	503	509	512	612	653	677	703	765	897	908

сливаемые серии имеют длину 2^k, тем не менее никаких явных мер предосторожности на случай таких исключений предусматривать не приходится! (См. упр. 8.) Проверки ступенек вниз в прежнем алгоритме заменены посредством уменьшения переменных q и r и проверки равенства нулю. Благодаря этому время выполнения на машине MIX асимптотически приближается к $11N \lg N$ машинным циклам, что несколько лучше результата, которого удалось достичь в алгоритме N.

На практике имеет смысл комбинировать алгоритм S с методом простых вставок. Вместо первых четырех просмотров алгоритма S можно методом простых вставок рассортировать группы, скажем, из 16 элементов, исключив таким образом довольно расточительные вспомогательные операции, связанные со слиянием коротких массивов. Как и в случае быстрой сортировки, такое комбинирование методов не влияет на асимптотическое время работы, но дает, тем не менее, немалую выгоду.

Рассмотрим теперь алгоритмы N и S с точки зрения выбора структур данных. Почему необходима память для $2N$, а не для N записей? Причина относительно проста: мы работаем с четырьмя списками переменного размера (два входных списка и два выходных списка в каждом просмотре); при этом для каждой пары последовательно распределенных списков используется стандартное понятие "совместный рост", обсуждавшееся в разделе 2.2.2. Но в любой момент времени половина памяти не используется, и после некоторого размышления становится ясно, что в действительности для наших четырех списков следовало бы воспользоваться *связным* распределением памяти. Если к каждой из N записей добавить поле связи, то все необходимое можно проделать, пользуясь алгоритмами слияния, которые выполняют простые манипуляции связями и совсем не перемещают сами записи. Добавление N полей связи, как правило, выгоднее добавления пространства памяти еще для N записей; отказавшись от перемещения записей, можно сэкономить и время. Итак, рассмотрим следующий алгоритм.

Алгоритм L (*Сортировка посредством слияния списков*). Предполагается, что в записях$R_1, \ldots, R_N$ содержатся ключи $K_1, \ldots, K_N$ и поля связи $L_1, \ldots, L_N$, в которых могут храниться числа от $-(N+1)$ до $(N+1)$. В начале и в конце массива имеются искусственные записи L_0 и L_{N+1} с полями связи R_0 и R_{N+1}. Этот алгоритм сортировки списков устанавливает поля связи таким образом, что записи оказываются связанными в порядке возрастания. После завершения сортировки L_0 указывает на запись с наименьшим ключом; при $1 \le k \le N$ связь L_k указывает на запись, следующую за R_k, или $L_k = 0$, если R_k — запись с наибольшим ключом (см. формулы 5.2.1–(13)).

В процессе работы этого алгоритма записи R_0 и R_{N+1} выполняют роли головных элементов двух линейных списков, подсписки которых в данный момент сливаются. Отрицательная связь означает конец подсписка, о котором известно, что он упорядочен; нулевая связь означает конец всего списка. Предполагается, что $N \geq 2$.

Через "$|L_s| \leftarrow p$" обозначена операция "Присвоить L_s значение p или $-p$, сохранив прежний знак L_s". Такая операция легко реализуется на компьютере MIX, но, к сожалению, не на большинстве других моделей компьютеров. Нетрудно изменить алгоритм, чтобы получить столь же эффективный метод для большинства других машин.

L1. [Подготовить два списка.] Установить $L_0 \leftarrow 1$, $L_{N+1} \leftarrow 2$, $L_i \leftarrow -(i+2)$ при $1 \leq i \leq N-2$ и $L_{N-1} \leftarrow L_N \leftarrow 0$. (Созданы два списка, содержащих записи $R_1, R_3, R_5, \ldots$ и $R_2, R_4, R_6, \ldots$ соответственно; отрицательные связи указывают, что каждый упорядоченный подсписок состоит всего лишь из одного элемента. Другой способ выполнения этого шага, если принять во внимание упорядоченность, которая могла присутствовать в исходных данных, рассмотрен в упр. 12.)

L2. [Начать новый просмотр.] Установить $s \leftarrow 0$, $t \leftarrow N+1$, $p \leftarrow L_s$, $q \leftarrow L_t$. Если $q = 0$, выполнение алгоритма завершается. (В процессе каждого просмотра p и q "пробегают" по спискам, которые подвергаются слиянию; s обычно указывает на последнюю обработанную запись текущего подсписка, а t — на конец только что выведенного подсписка.)

L3. [Сравнение $K_p : K_q$.] Если $K_p > K_q$, перейти к шагу L6.

L4. [Продвинуть p.] Установить $|L_s| \leftarrow p$, $s \leftarrow p$, $p \leftarrow L_p$. Если $p > 0$, вернуться к шагу L3.

L5. [Завершить подсписок.] Установить $L_s \leftarrow q$, $s \leftarrow t$. Далее установить $t \leftarrow q$ и $q \leftarrow L_q$ один или более раз, пока не станет $q \leq 0$, после чего перейти к шагу L8.

L6. [Продвинуть q.] (Шаги L6 и L7 двойственны шагам L4 и L5.) Установить $|L_s| \leftarrow q$, $s \leftarrow q$, $q \leftarrow L_q$. Если $q > 0$, вернуться к шагу L3.

L7. [Завершить подсписок.] Установить $L_s \leftarrow p$, $s \leftarrow t$. Затем установить $t \leftarrow p$ и $p \leftarrow L_p$ один или более раз, пока не станет $p \leq 0$.

L8. [Конец просмотра?] (К этому моменту $p \leq 0$ и $q \leq 0$, так как оба указателя продвинулись до конца соответствующих подсписков.) Установить $p \leftarrow -p$, $q \leftarrow -q$. Если $q = 0$, установить $|L_s| \leftarrow p$, $|L_t| \leftarrow 0$ и вернуться к шагу L2. В противном случае вернуться к шагу L3. ▌

Пример работы этого алгоритма приведен в табл. 3, в которой показаны связи к моменту выполнения шага L2. По окончании выполнения алгоритма можно, пользуясь методом из упр. 5.2–12, перекомпоновать в памяти записи $R_1, \ldots, R_N$ так, чтобы их ключи стали упорядоченными. Нужно отметить интересную аналогию между слиянием списков и сложением разреженных многочленов (см. алгоритм 2.2.4A).

Напишем теперь MIX-программу для алгоритма L, чтобы выяснить, столь ли выгодно оперировать списками с точки зрения скорости выполнения, как и с точки зрения расхода памяти?

Программа L (*Сортировка посредством слияния списков*). Для удобства предполагается, что записи занимают одно слово, причем L_j хранится в поле $(0:2)$ и

Таблица 3

СОРТИРОВКА ПОСРЕДСТВОМ СЛИЯНИЯ СПИСКОВ

j	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
K_j	–	503	087	512	061	908	170	897	275	653	426	154	509	612	677	765	703	–
L_j	1	−3	−4	−5	−6	−7	−8	−9	−10	−11	−12	−13	−14	−15	−16	0	0	2
L_j	2	−6	1	−8	3	−10	5	−11	7	−13	9	12	−16	14	0	0	15	4
L_j	4	3	1	−11	2	−13	8	5	7	0	12	10	9	14	16	0	15	6
L_j	4	3	6	7	2	0	8	5	1	14	12	10	13	9	16	0	15	11
L_j	4	12	11	13	2	0	8	5	10	14	1	6	3	9	16	7	15	0

K_j — в поле (3 : 5) по адресу INPUT + j. Состояние регистров таково: rI1 $\equiv p$, rI2 $\equiv q$, rI3 $\equiv s$, rI4 $\equiv t$, rA $\equiv K_q$; $N \geq 2$.

```
01 L      EQU  0:2                    Определение имен полей.
02 ABSL   EQU  1:2
03 KEY    EQU  3:5
04 START  ENT1 N-2                1     L1. Подготовить два списка.
05        ENNA 2,1                N-2
06        STA  INPUT,1(L)         N-2   L_i ← -(i+2).
07        DEC1 1                  N-2
08        J1P  *-3                N-2   N-2 ≥ i > 0.
09        ENTA 1                  1
10        STA  INPUT(L)           1     L_0 ← 1.
11        ENTA 2                  1
12        STA  INPUT+N+1(L)       1     L_{N+1} ← 2.
13        STZ  INPUT+N-1(L)       1     L_{N-1} ← 0.
14        STZ  INPUT+N(L)         1     L_N ← 0.
15        JMP  L2                 1     Перейти к шагу L2.
16 L3Q    LDA  INPUT,2            C''+B' L3. Сравнение K_p : K_q.
17 L3P    CMPA INPUT,1(KEY)       C
18        JL   L6                 C     Перейти к шагу L6, если K_q < K_p.
19 L4     ST1  INPUT,3(ABSL)      C'    L4. Продвинуть p. |L_s| ← p.
20        ENT3 0,1                C'    s ← p.
21        LD1  INPUT,1(L)         C'    p ← L_p.
22        J1P  L3P                C'    Перейти к шагу L3, если p > 0.
23 L5     ST2  INPUT,3(L)         B'    L5. Завершить подсписок. L_s ← q.
24        ENT3 0,4                B'    s ← t.
25        ENT4 0,2                D'    t ← q.
26        LD2  INPUT,2(L)         D'    q ← L_q.
27        J2P  *-2                D'    Повторить, если q > 0.
28        JMP  L8                 B'    Перейти к шагу L8.
29 L6     ST2  INPUT,3(ABSL)      C''   L6. Продвинуть q. |L_s| ← q.
30        ENT3 0,2                C''   s ← q.
31        LD2  INPUT,2(L)         C''   q ← L_q.
32        J2P  L3Q                C''   Перейти к шагу L3, если q > 0.
33 L7     ST1  INPUT,3(L)         B''   L7. Завершить подсписок. L_s ← p.
34        ENT3 0,4                B''   s ← t.
35        ENT4 0,1                D''   t ← p.
36        LD1  INPUT,1(L)         D''   p ← L_p.
37        J1P  *-2                D''   Повторить, если p > 0.
38 L8     ENN1 0,1                B     L8. Конец просмотра? p ← -p.
```

```
39          ENN2 0,2              B      q ← −q.
40          J2NZ L3Q              B      Перейти к шагу L3, если q ≠ 0.
41          ST1  INPUT,3(ABSL)    A      |L_s| ← p.
42          STZ  INPUT,4(ABSL)    A      |L_t| ← 0.
43  L2      ENT3 0                A+1    L2. Начать новый просмотр. s ← 0.
44          ENT4 N+1              A+1    t ← N + 1.
45          LD1  INPUT(L)         A+1    p ← L_s.
46          LD2  INPUT+N+1(L)     A+1    q ← L_t.
47          J2NZ L3Q              A+1    Перейти к шагу L3, если q ≠ 0.  ▌
```

Время выполнения этой программы можно оценить при помощи методов, которыми мы уже не раз пользовались (см. упр. 13 и 14). В среднем оно равно приблизительно $(10N \lg N + 4.92N)$ машинных циклов с небольшим стандартным отклонением порядка $\sqrt{N}$. В упр. 15 показано, что за счет некоторого удлинения программы можно сократить время примерно до $9N \lg N$.

Итак, в случае внутреннего слияния связное распределение памяти имеет бесспорные преимущества перед последовательным распределением: требуется меньше памяти и программа работает на 10–20% быстрее. Аналогичные алгоритмы опубликованы в работах L. J. Woodrum, *IBM Systems J.* **8** (1969), 189–203, и A. D. Woodall, *Comp. J.* **13** (1970), 110–111.

УПРАЖНЕНИЯ

1. [*21*] Обобщите алгоритм M для *k-путевого слияния* исходных массивов $x_{i1} \le \cdots \le x_{im_i}$ при $i = 1, 2, \ldots, k$.

2. [*M24*] Считая, что все $\binom{m+n}{m}$ возможных расположений m элементов x среди n элементов y равновероятны, найдите математическое ожидание и стандартное отклонение числа выполнений шага M2 в алгоритме M. Чему равны максимальное и минимальное значения этой величины?

▸ **3.** [*20*] (*Обновление.*) Даны записи $R_1, \ldots, R_M$ и $R'_1, \ldots, R'_N$, ключи которых различны и упорядочены, т. е. $K_1 < \cdots < K_M$ и $K'_1 < \cdots < K'_N$. Как нужно изменить алгоритм M, чтобы в результате слияния получался массив, в котором *отсутствуют* записи R_i первого массива, если во втором массиве также есть запись с таким же ключом?

4. [*21*] В основном тексте раздела отмечено, что сортировку методом слияния можно рассматривать как обобщение сортировки методом вставок. Покажите, что метод слияний имеет непосредственное отношение и к выбору из дерева (воспользуйтесь в качестве иллюстрации рис. 23).

▸ **5.** [*21*] Докажите, что переменные i и j, используемые при выполнении шагов N6 и N10, не могут быть равны. (Поэтому на данных шагах проверка на случай возможного перехода к N13 необязательна.)

6. [*22*] Найдите такую перестановку $K_1 K_2 \ldots K_{16}$ множества $\{1, 2, \ldots, 16\}$, что

$$K_2 > K_3, \quad K_4 > K_5, \quad K_6 > K_7, \quad K_8 > K_9, \quad K_{10} < K_{11}, \quad K_{12} < K_{13}, \quad K_{14} < K_{15},$$

которая, тем не менее, будет рассортирована при помощи алгоритма N всего за два просмотра. (Так как в искомой перестановке имеется не менее восьми серий, можно было бы ожидать, что после первого просмотра останется по меньшей мере четыре серии, после второго просмотра — две серии и сортировка, как правило, не завершится раньше третьего просмотра. Как можно обойтись всего двумя просмотрами?)

7. [*16*] Найдите точную формулу, выражающую число проходов алгоритма S в виде функции от N.

8. [*22*] Предполагается, что во время выполнения алгоритма переменные q и r представляют длины неслитых частей серий, обрабатываемых в данный момент; в начале работы как q, так и r устанавливаются равными p, в то время как серии не всегда имеют такую длину. Почему же алгоритм, тем не менее, работает правильно?

9. [*24*] Напишите MIX-программу для алгоритма S. Выразите частоту выполнения каждой команды через параметры, подобные $A, B', B'', C', \ldots$ в программе L.

10. [*25*] (Д. А. Белл (D. A. Bell).) Покажите, что простое двухпутевое слияние массивов с последовательно расположенными элементами можно выполнить, имея всего $\frac{3}{2}N$ ячеек памяти, а не $2N$, как в алгоритме S.

11. [*21*] Является ли алгоритм L алгоритмом устойчивой сортировки?

▸ **12.** [*22*] Измените шаг L1 алгоритма L так, чтобы двухпутевое слияние стало "естественным", используя наличие восходящих серий в исходном массиве. (В частности, если исходные данные уже упорядочены, то на шаге L2 можно завершить выполнение алгоритма сразу же после выполнения нового варианта шага L1.)

▸ **13.** [*M34*] Проанализируйте среднее время работы программы L так, как были проанализированы другие алгоритмы в этой главе. Дайте толкование параметрам $A, B, B', \ldots$ и объясните, как вычислить их точные средние значения. Сколько времени затратит программа L на сортировку 16 чисел в табл. 3?

14. [*M24*] Пусть двоичное представление числа N есть $2^{e_1} + 2^{e_2} + \cdots + 2^{e_t}$, где $e_1 > e_2 > \cdots > e_t \geq 0$, $t \geq 1$. Докажите, что максимальное число сравнений ключей, выполняемых алгоритмом L, равно $1 - 2^{e_t} + \sum_{k=1}^{t}(e_k + k - 1)2^{e_k}$.

15. [*20*] Если промоделировать вручную работу алгоритма L, то обнаружится, что в нем иногда выполняются лишние операции. Примерно в половине случаев не нужны присваивания $|L_s| \leftarrow p$, $|L_s| \leftarrow q$ на шагах L4 и L6, поскольку мы имеем $L_s = p$ (или q) всякий раз, когда возвращаемся из шага L4 (или L6) к шагу L3. Как улучшить программу L и избавиться от этих лишних присваиваний?

16. [*28*] Разработайте алгоритм слияния списков, подобный алгоритму L, но основанный на трехпутевом слиянии.

17. [*20*] (Дж. Мак-Карти (J. McCarthy).) Пусть двоичное представление числа N такое же, как в упр. 14, и предположим, что дано N записей, организованных в t упорядоченных подмассивов, имеющих размеры $2^{e_1}, 2^{e_2}, \ldots, 2^{e_t}$ соответственно. Покажите, как можно сохранить такое состояние при добавлении $((N+1)$-й записи и $N \leftarrow N+1$. (Полученный алгоритм можно назвать *оперативной сортировкой методом слияния.*)

18. [*40*] (М. А. Кронрод.) Можно ли рассортировать массив из N записей, содержащий всего две серии,

$$K_1 \leq \cdots \leq K_M \qquad \text{и} \qquad K_{M+1} \leq \cdots \leq K_N,$$

за $O(N)$ операций в оперативной памяти, используя *лишь небольшой дополнительный участок*, размер которого не зависит от M и N? (Все алгоритмы слияния, описанные в этом разделе, используют дополнительную память, размер которой пропорционален N.)

19. [*26*] Рассмотрим железнодорожную сортировочную станцию с n накопительными тупиками — аналогами стека. На рис. 31 показана такая станция при $n = 5$. Операции в сети путей подобного типа имеют некоторое отношение к алгоритмам сортировки с n проходами. В упр. с 2.2.1–2 по 2.2.1–5 была рассмотрена такого рода железнодорожная сеть с одним тупиком. Было показано, что если с правого конца поступает N вагонов, то слева

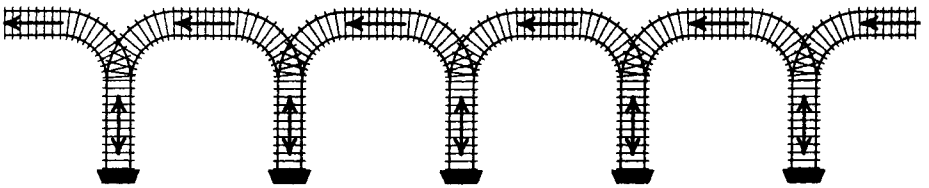


Рис. 31. Сеть железнодорожных путей с пятью тупиками.

может появиться сравнительно небольшое количество из $N!$ всевозможных перестановок этих вагонов.

Предположим, что в n-тупиковый разъезд справа заталкивается 2^n вагонов. Докажите, что при помощи подходящей последовательности операций можно получить слева любую из $2^n!$ всевозможных перестановок этих вагонов. (Каждый тупик достаточно велик, и при необходимости в него можно поместить все вагоны.)

20. [*47*] В обозначениях упр. 2.2.1–4 при помощи железнодорожной сети с n тупиками можно получить не более a_N^n перестановок N элементов. Следовательно, для получения всех $N!$ перестановок требуется не менее $\log N!/\log a_N \approx \log_4 N$ тупиков. В упр. 19 показано, что нужно не более $\lceil \lg N \rceil$ тупиков. Какова истинная скорость роста необходимого числа тупиков при $N \to \infty$?

21. [*23*] (А. Дж. Смит (A. J. Smith).) Объясните, как можно обобщить алгоритм L, чтобы он не только выполнял сортировку, но и вычислял число *инверсий* в исходной перестановке.

22. [*28*] (Дж. К. Р. Барнет (J. K. R. Barnett).) Разработайте способ повышения скорости сортировки методом слияния для многословных ключей. (В упр. 5.2.2–30 рассматривается аналогичная проблема применительно к быстрой сортировке.)

23. [*M30*] В упр. 13 и 14 анализируется "восходящая", или итеративная, версия сортировки методом слияния, где параметр цены $c(N)$ сортировки N элементов удовлетворяет рекуррентному соотношению

$$c(N) = c(2^k) + c(N - 2^k) + f(2^k, N - 2^k) \qquad \text{при } 2^k < N \le 2^{k+1}$$

и $f(m, n)$ есть цена слияния m элементов с n. Проанализируйте "нисходящую" версию или рекуррентное соотношение наподобие "разделяй и властвуй",

$$c(N) = c(\lceil N/2 \rceil) + c(\lfloor N/2 \rfloor) + f(\lceil N/2 \rceil, \lfloor N/2 \rfloor) \qquad \text{при } N > 1,$$

которое имеет место при использовании рекуррентной программы сортировки.

5.2.5. Сортировка методом распределения

Рассмотрим теперь интересный класс методов сортировки, который, как показано в разделе 5.4.7, по своей сути является *обратным* слиянию. Читателям, знакомым с перфокарточным оборудованием, хорошо известна эффективная процедура, применяемая в машинах для сортировки карт задолго до появления электронных компьютеров. Ту же идею можно использовать и для программирования компьютеров. Она общеизвестна под названиями "поразрядная сортировка", "карманная сортировка" (bucket sorting) и "цифровая сортировка" (digital sorting), поскольку базируется на анализе цифр ключей.

Предположим, необходимо рассортировать колоду из 52 игральных карт. Определим упорядочение по старшинству (достоинству) карт в масти

$$\mathtt{A} < 2 < 3 < 4 < 5 < 6 < 7 < 8 < 9 < 10 < \mathtt{J} < \mathtt{Q} < \mathtt{K},$$

а также по масти

$$\clubsuit < \diamondsuit < \heartsuit < \spadesuit.$$

Одна карта предшествует другой, если либо (i) она младше по масти, либо (ii) масти обеих карт одинаковы, но она младше по достоинству. (Это частный случай *лексикографического упорядочения* на множестве упорядоченных пар предметов; см. упр. 5–2.) Таким образом, получаем

$$\mathtt{A}\clubsuit < 2\clubsuit < \cdots < \mathtt{K}\clubsuit < \mathtt{A}\diamondsuit < \cdots < \mathtt{Q}\spadesuit < \mathtt{K}\spadesuit.$$

Можно было бы рассортировать карты одним из обсуждавшихся ранее методов. Люди, как правило, пользуются способом, по сути, аналогичным обменной поразрядной сортировке. Естественно рассортировывать карты сначала по мастям, разложив их в четыре стопки, а затем перекладывать карты внутри каждой стопки до тех пор, пока они не будут упорядочены по достоинству.

Но существует более быстрый способ! Сначала разложим карты в 13 стопок лицевой стороной вверх по их достоинству. После этого соберем все стопки вместе: снизу — тузы, затем — двойки, тройки и т. д. и сверху — короли (лицевой стороной вверх). Далее перевернем колоду рубашками вверх и снова разложим ее, на этот раз в четыре стопки по масти. Сложив вместе полученные стопки так, чтобы внизу были трефы, затем бубны, черви и пики, получим упорядоченную колоду карт.

Та же идея годится для сортировки числовых и буквенных данных. Почему же она работает — приводит к верному результату? Потому что (в нашем примере с игральными картами), если две карты при последнем раскладе попали в разные стопки, то они имеют разные масти, так что карта с меньшей мастью младше. Если же карты одной масти, они уже находятся в нужном порядке вследствие предварительной сортировки. Иначе говоря, при втором раскладе в каждой из четырех стопок карты будут расположены в порядке возрастания. Это рассуждение можно распространить на более общий случай и показать, что таким способом можно рассортировать любое множество с лексикографическим упорядочением (подробности приводятся в упр. 5–2, в начале этой главы).

Только что описанный метод сортировки сразу не очевиден, и не ясно, кто же первый обнаружил, что он настолько удобен. В брошюре "The Inventory Simplified" (Изобретательство — это очень просто), опубликованной отделением Tabulating Machines Company фирмы IBM в 1923 году, на 19 страницах представлен интересный цифровой метод вычисления сумм произведений на электрической сортировальной машине этой фирмы. Пусть, например, нужно перемножить два числа, пробитых соответственно в колонках 1–10 и 23–25, и вычислить сумму таких произведений для большого числа карт. Сначала можно рассортировать карты по колонке 25 и найти при помощи табулятора величины $a_1, a_2, \ldots, a_9$, где a_k — сумма чисел из колонок 1–10 по всем картам, на которых в колонке 25 пробита цифра k. Затем можно рассортировать карты по колонке 24 и найти аналогичные суммы $b_1, b_2, \ldots, b_9$ и по колонке 23, получив $c_1, c_2, \ldots, c_9$. Легко видеть, что искомая сумма произведений равна

$$a_1 + 2a_2 + \cdots + 9a_9 + 10b_1 + 20b_2 + \cdots + 90b_9 + 100c_1 + 200c_2 + \cdots + 900c_9.$$

Такой перфокарточный метод табуляции естественным образом привел к идее поразрядной сортировки "сначала по младшей цифре", так что, по-видимому, она впервые

стала известна операторам этих машин. Первое опубликованное упоминание об этом методе содержится в ранней работе Л. Дж. Комри (L. J. Comrie), посвященной конструированию перфокарточного оборудования [*Transactions of the Office Machinery Users' Assoc., Ltd.* (1929), 25–37, особенно с. 28].

Чтобы выполнить поразрядную сортировку с помощью электронного компьютера, необходимо решить, как представлять стопки. Пусть имеется M стопок; можно было бы выделить M областей памяти и переслать каждую исходную запись в соответствующую область. Но это решение нас не удовлетворяет, потому что в каждой области должно быть достаточно места для хранения N элементов, и тогда потребуется память для $(M+1)N$ записей. Такая чрезмерная потребность в памяти заставляла большинство программистов отказываться от применения поразрядной сортировки при программировании компьютеров до тех пор, пока Х. Х. Сьюворд (H. H. Seward) в Master's thesis, M. I. T. Digital Computer Laboratory Report R-232 (1954), 25–28, не показал, что того же эффекта можно достичь, используя память всего для $2N$ записей и M счетчиков. Сделав один предварительный просмотр данных, можно просто посчитать, сколько элементов попадет в каждую область, а это позволит точно распределить память для стопок. Мы уже применяли эту идею при распределяющей сортировке (алгоритм 5.2D).

Итак, поразрядную сортировку можно выполнить следующим образом: сначала выполнить распределяющую сортировку по *младшим цифрам ключей* (в системе счисления с основанием M) и переместить при этом записи из области ввода во вспомогательную область; затем осуществить еще одну распределяющую сортировку по следующей цифре, переместив на сей раз записи обратно в исходную область, и так до тех пор, пока после завершающего просмотра (сортировки по старшей цифре) все ключи не окажутся расположенными в нужном порядке.

Если имеется десятичная машина, а ключи — 12-разрядные числа и если N весьма велико, то можно выбрать $M = 1000$ (считая три десятичные цифры за одну в системе счисления с основанием 1 000). Независимо от величины N сортировка будет выполнена за четыре просмотра. Аналогично, если имеется двоичная машина, а ключи — 40-разрядные двоичные числа, то можно положить, что $M = 1024 = 2^{10}$, и также завершить сортировку за четыре просмотра. Фактически каждый просмотр состоит из трех групп операций (подсчет, распределение памяти, перезапись). Э. Г. Френд (E. H. Friend) [*JACM* **3** (1956), 151] предложил комбинировать два из этих трех действий, заплатив за это добавлением еще M ячеек: накапливать значения счетчиков для $(k+1)$-го просмотра одновременно с перезаписью при k-м просмотре.

В табл. 1 показано применение поразрядной сортировки к нашим 16 ключам при $M = 10$. При таких малых N поразрядная сортировка, как правило, не особенно эффективна, так что этот маленький пример предназначен, главным образом, для того, чтобы продемонстрировать достаточность метода, а не его эффективность.

Искушенный современный читатель заметит, однако, что идея использования счетчиков для распределения памяти привязана к старомодным понятиям о последовательном выделении памяти и соответственно о последовательном размещении данных в памяти. Нам же известно, что специально для работы с множеством таблиц переменной длины придумано связное распределение. Поэтому для поразрядной сортировки естественно будет воспользоваться связными структурами данных.

Таблица 1

ПОРАЗРЯДНАЯ СОРТИРОВКА

Область ввода:	503	087	512	061	908	170	897	275	653	426	154	509	612	677	765	703
Счетчики для цифр единиц:							1	1	2	3	1	2	1	3	1	1
Распределение памяти соответственно этим счетчикам:							1	2	4	7	8	10	11	14	15	16
Вспомогательная область:	170	061	512	612	503	653	703	154	275	765	426	087	897	677	908	509
Счетчики для цифр десятков:							4	2	1	0	0	2	2	3	1	1
Распределение памяти соответственно этим счетчикам:							4	6	7	7	7	9	11	14	15	16
Область ввода:	503	703	908	509	512	612	426	653	154	061	765	170	275	677	087	897
Счетчики для цифр сотен:							2	2	1	0	1	3	3	2	1	1
Распределение памяти соответственно этим счетчикам:							2	4	5	5	6	9	12	14	15	16
Вспомогательная область:	061	087	154	170	275	426	503	509	512	612	653	677	703	765	897	908

Поскольку каждая стопка просматривается последовательно, все, что нам нужно, — иметь при каждом элементе одну-единственную ссылку на следующий элемент. Кроме того, никогда не придется перемещать записи: достаточно откорректировать связи — и можно смело двигаться дальше по спискам. Объем необходимой памяти равен $(1+\epsilon)N+2\epsilon M$ записей, где ϵ — пространство, занимаемое одним полем связи. Довольно интересны формальные подробности этой процедуры, поскольку они дают прекрасный пример типовых операций со структурами данных, объединяющих в себе работу с последовательным и связным представлениями данных в памяти.

Алгоритм R (*Поразрядная сортировка списка*). Предполагается (рис. 32), что в каждой из записей $R_1, \ldots, R_N$ содержится поле связи LINK, ключи представляют собой последовательность из p элементов

$$(a_1, a_2, \ldots, a_p), \qquad 0 \le a_i < M, \tag{1}$$

и отношение порядка — лексикографическое, т. е.

$$(a_1, a_2, \ldots, a_p) < (b_1, b_2, \ldots, b_p) \tag{2}$$

тогда и только тогда, когда существует такой индекс j, $1 \le j \le p$, что

$$a_i = b_i \quad \text{для всех } i < j, \qquad \text{но} \qquad a_j < b_j. \tag{3}$$

Ключи можно представлять, в частности, как числа, записанные в системе счисления с основанием M:

$$a_1 M^{p-1} + a_2 M^{p-2} \cdots + a_{p-1} M + a_p. \tag{4}$$

В этом случае лексикографическое отношение порядка соответствует обычному упорядочению множества неотрицательных чисел. Ключи также могут быть цепочками букв алфавита и т. д.

Во время сортировки формируются M "стопок", как в сортировальной машине для перфокарт. Стопки фактически представляют собой очереди в смысле главы 2, поскольку мы связываем их вместе таким образом, что они всегда просматриваются по принципу "первым пришел — первым вышел". Для каждой стопки имеются

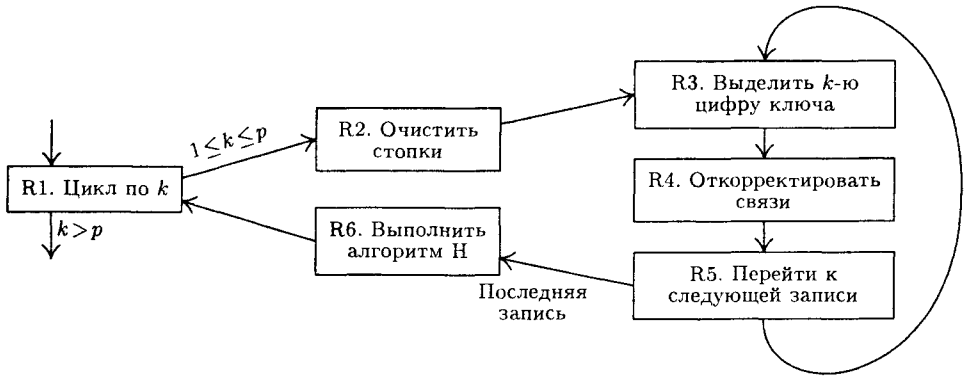


Рис. 32. Поразрядная сортировка списка.

две переменные-указатели: TOP[i] и BOTM[i], $0 \le i < M$, и, как и в главе 2, предполагается, что

$$\texttt{LINK(LOC(BOTM[}i\texttt{]))} \equiv \texttt{BOTM[}i\texttt{]}. \tag{5}$$

R1. [Цикл по k.] Вначале установить P $\leftarrow$ LOC(R_N), указатель на последнюю запись. Затем выполнить шаги R2–R6 при $k = 1, 2, \ldots, p$ (шаги R2–R6 составляют один "проход" — просмотр исходных данных) и завершить выполнение алгоритма. Переменная P будет указывать на запись с наименьшим ключом, LINK(P) — на запись со следующим по величине ключом, LINK(LINK(P)) — на следующую запись и т. д. Поле LINK последней записи будет равно Λ.

R2. [Очистить стопки.] При $0 \le i < M$ установить TOP[i] $\leftarrow$ LOC(BOTM[i]) и BOTM[i] $\leftarrow \Lambda$.

R3. [Выделить k-ю цифру ключа.] Пусть KEY(P) — ключ записи, на которую указывает P, — равен $(a_1, a_2, \ldots, a_p)$. Установить $i \leftarrow a_{p+1-k}$, k-ю младшую цифру этого ключа.

R4. [Откорректировать связи.] Установить LINK(TOP[i]) $\leftarrow$ P и TOP[i] $\leftarrow$ P.

R5. [Перейти к следующей записи.] Если $k = 1$ (первый просмотр) и если P = LOC(R_j) при некотором $j \ne 1$, установить P $\leftarrow$ LOC(R_{j-1}) и возвратиться к шагу R3. Если $k > 1$ (не первый просмотр), то установить P $\leftarrow$ LINK(P) и возвратиться к R3, если P $\ne \Lambda$.

R6. [Выполнить алгоритм H.] (Теперь все элементы распределены по стопкам.) Выполнить приведенный ниже алгоритм H, который "сцепляет" отдельные стопки в один список, подготавливая их к следующему просмотру. Установить P $\leftarrow$ BOTM[0], указатель на первый элемент объединенного списка (см. упр. 3). ▌

Алгоритм H (*Сцепление очередей*). Из M данных очередей со связями, удовлетворяющими соглашениям алгоритма R, данный алгоритм создает одну очередь, меняя при этом не более M связей. В результате BOTM[0] указывает на первый элемент и стопка 0 предшествует стопке 1 ..., стопка $M-2$ предшествует стопке $M-1$.

H1. [Начальная установка.] Установить $i \leftarrow 0$.

H2. [Указатель на вершину стопки.] Установить P $\leftarrow$ TOP[i].

H3. [Следующая стопка.] Увеличить i на 1. Если $i = M$, то установить LINK(P) $\leftarrow \Lambda$ и завершить выполнение алгоритма.

H4. [Стопка пуста?] Если BOTM[i] $= \Lambda$, возвратиться к шагу H3.

H5. [Сцепить стопки.] Установить LINK(P) $\leftarrow$ BOTM[i]. Возвратиться к шагу H2. ▮

На рис. 33 показано содержимое стопок после каждого из трех просмотров, выполняемых при сортировке наших 16 чисел с $M = 10$. Алгоритм R очень просто запрограммировать в системе команд MIX, если найти удобный способ изменения операции на шагах R3 и R5 от одного просмотра к другому. В следующей программе этого удалось достичь, не жертвуя скоростью внутреннего цикла, путем предварительной записи двух команд в тело программы. Заметим, что TOP[i] и BOTM[i] можно упаковать в одно слово.

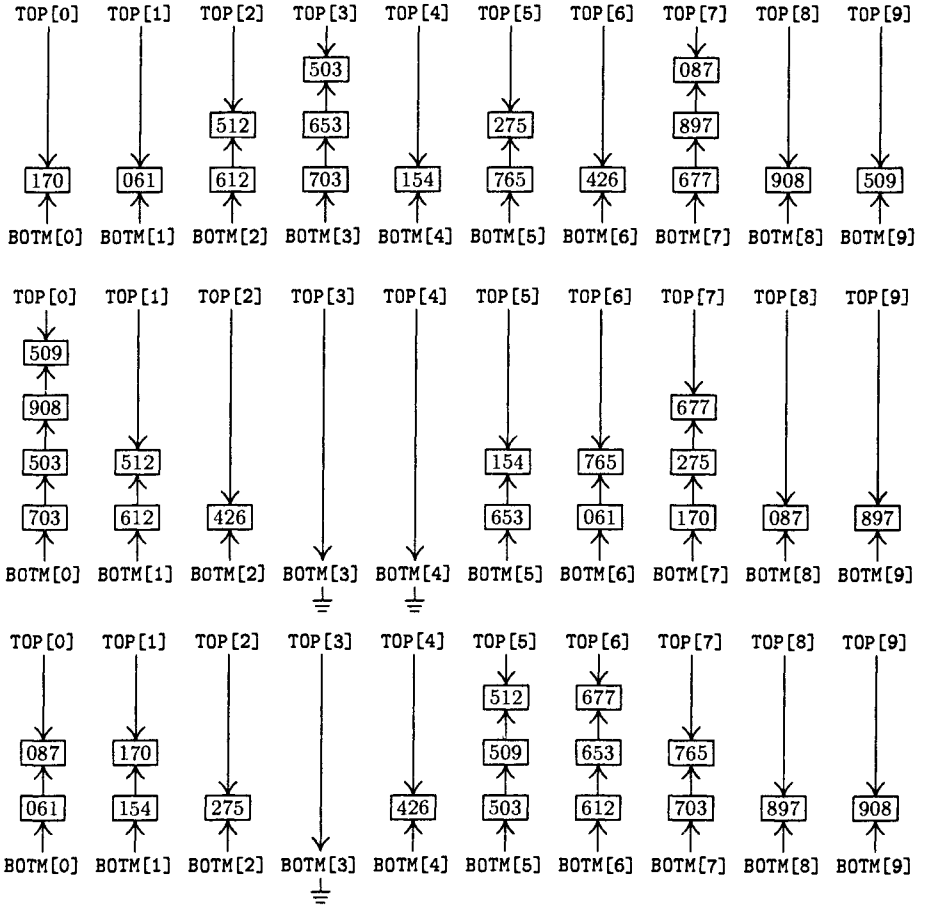


Рис. 33. Поразрядная сортировка с использованием связного распределения памяти (показано содержимое всех десяти стопок после каждого просмотра).

Программа R (*Поразрядная сортировка списков*). Предполагается, что исходные ключи в ячейках по адресу от INPUT+1 до INPUT+N содержат $p = 3$ компоненты

(a_1, a_2, a_3), хранящиеся в полях (1:1), (2:2) и (3:3) соответственно. (Таким образом, считается, что значение M меньше или равно размеру байта компьютера MIX.) В поле (4:5) каждой записи хранится связь LINK. Пусть TOP[i] $\equiv$ PILES + i(1:2) и BOTM[i] $\equiv$ PILES + i(4:5) при $0 \le i < M$. Удобно указывать в поле связи положение относительно адреса INPUT, так что LOC(BOTM[i]) = PILES + i − INPUT. Чтобы избежать появления отрицательных связей, нужно расположить таблицу PILES после таблицы INPUT. Значения индексных регистров таковы: rI1 $\equiv$ P, rI2 $\equiv i$, rI3 $\equiv 3 - k$, rI4 $\equiv$ TOP[i]. Во время выполнения алгоритма H rI2 $\equiv i - M$.

```
01  LINK   EQU  4:5
02  TOP    EQU  1:2
03  START  ENT1 N                    1      R1. Цикл по k. P = LOC(R_N).
04         ENT3 2                    1      k ← 1.
05  2H     ENT2 M-1                  3      R2. Очистить стопки.
06         ENTA PILES-INPUT,2       3M      LOC(BOTM[i])
07         STA  PILES,2(TOP)        3M          → TOP[i]
08         STZ  PILES,2(LINK)       3M      BOTM[i] ← Λ.
09         DEC2 1                   3M
10         J2NN *-4                 3M      M > i ≥ 0.
11         LDA  R3SW,3               3
12         STA  3F                   3      Изменить команды на проходе k.
13         LDA  R5SW,3               3
14         STA  5F                   3
15  3H    [LD2  INPUT,1(3:3)]               R3. Извлечь k-ю цифру ключа.
16  4H     LD4  PILES,2(TOP)        3N      R4. Откорректировать связи.
17         ST1  INPUT,4(LINK)       3N      LINK(TOP[i]) ← P.
18         ST1  PILES,2(TOP)        3N      TOP[i] ← P.
19  5H    [DEC1 1]                          R5. Перейти к следующей записи.
20         J1NZ 3B                  3N      Перейти к шагу R3, если конец прохода.
21  6H     ENN2 M                    3      R6. Выполнить алгоритм H.
22         JMP  7F                   3      Перейти к шагу H2, если i ← 0.
23  R3SW   LD2  INPUT,1(1:1)         N      Команды для шага R3, если k = 3.
24         LD2  INPUT,1(2:2)         N      Команды для шага R3, если k = 2.
25         LD2  INPUT,1(3:3)         N      Команды для шага R3, если k = 1.
26  R5SW   LD1  INPUT,1(LINK)        N      Команды для шага R5, если k = 3.
27         LD1  INPUT,1(LINK)        N      Команды для шага R5, если k = 2.
28         DEC1 1                    N      Команды для шага R5, если k = 1.
29  9H     LDA  PILES+M,2(LINK)  3M−3       H4. Стопка пуста?
30         JAZ  8F               3M−3       Перейти к шагу H3, если BOTM[i] = Λ.
31         STA  INPUT,1(LINK)  3M−3−E       H5. Сцепить стопки.
32  7H     LD1  PILES+M,2(TOP)  3M − E      H2. Указатель на вершину стопки.
33  8H     INC2 1                   3M      H3. Следующая стопка. i ← i + 1.
34         J2NZ 9B                  3M      Перейти к шагу H4, если i ≠ M.
35         STZ  INPUT,1(LINK)        3      LINK(P) ← Λ.
36         LD1  PILES(LINK)          3      P ← BOTM[0].
37         DEC3 1                    3
38         J3NN 2B                   3      Цикл для 1 ≤ k ≤ 3.  ▌
```

Время выполнения программы R равно $32N + 48M + 38 - 4E$, где N — число исходных записей, M — основание системы счисления (число стопок), а E — число встретившихся пустых стопок. Сравнение с другими программами, построенными

на основе аналогичных предположений (программы 5.2.1M и 5.2.4L), говорит явно в пользу программы R. Время выполнения p-проходного варианта программы R равно $(11p-1)N+O(pM)$ машинных циклов; критический фактор, влияющий на время выполнения, — внутренний цикл, который содержит пять обращений к памяти и один переход. Для типичного компьютера $M=b^r$ и $p=\lceil t/r\rceil$, где t — число цифр в ключах, представленных в системе счисления с основанием b. С ростом r убывает p, так что можно воспользоваться нашими формулами для определения "наилучшего" значения r.

Единственная переменная величина в формуле для времени выполнения — это E (число пустых стопок, обнаруженных на шаге H4). Предположим, что все M^N последовательностей цифр M-ичной системы счисления равновероятны. При анализе "покер-теста" в разделе 3.3.2D было показано, как вычислять вероятность того, что при каждом просмотре встретится ровно $M-r$ пустых стопок. На каждом проходе она равна

$$\frac{M(M-1)\ldots(M-r+1)}{M^N}\left\{{N \atop r}\right\}, \tag{6}$$

где $\left\{{N \atop r}\right\}$ — число Стирлинга второго рода. Согласно упр. 5

$$E=\left(\min\ \max(M-N,0)p,\ \text{ave}\ M\left(1-\frac{1}{M}\right)^N p,\ \max\ (M-1)p\right). \tag{7}$$

В последние годы появляется все больше компьютеров с конвейерной (pipeline) и магистральной (number-crunching) архитектурами. Эти машины имеют несколько арифметических устройств и схему "опережения", так что обращения к памяти и вычисления могут в значительной степени совмещаться во времени. Но эффективность таких компьютеров заметно снижается при наличии условных переходов, если только эти переходы не происходят почти всегда в одном и том же направлении. Внутренний цикл поразрядной сортировки хорошо приспособлен для таких машин, поскольку это простое итеративное вычисление — типичное "пережевывание чисел". Поэтому *для компьютеров с магистральной архитектурой метод поразрядной сортировки обычно является наиболее эффективным из всех известных методов внутренней сортировки* при условии, что N не слишком мало́ и ключи не слишком длинные.

Разумеется, если ключи уж очень длинные, поразрядная сортировка не так эффективна. Представьте себе, например, что нужно рассортировать 60-разрядные десятичные числа за 20 проходов поразрядной сортировки, используя $M=10^3$. Очень мало пар чисел будет иметь одинаковые ключи в первых 9 разрядах, так что первые 17 проходов выполнятся почти впустую. При анализе обменной поразрядной сортировки мы обнаружили, что вовсе необязательно проверять много битов ключей, если просматривать их не справа налево, а слева направо. Поэтому давайте возвратимся к поразрядной сортировке, при которой ключи просматриваются, начиная со старших цифр (СЦ), а не с младших цифр (МЦ).

Мы уже отмечали, что идея СЦ-поразрядной сортировки приходит в голову естественным образом. В самом деле, нетрудно понять, почему при сортировке почты в отделениях связи пользуются именно этим методом. Большое количество писем можно рассортировать по отдельным мешкам, соответствующим географическим областям. Теперь в каждом мешке содержится меньше писем, которые

можно независимо рассортировать по другим мешкам, соответствующим все более мелким районам. (Разумеется, прежде чем продолжить сортировку писем, их можно переправить поближе к месту назначения.) Принцип "разделяй и властвуй" весьма привлекателен, и единственная причина его непригодности для сортировки перфокарт состоит в том, что большое количество стопок приводит к путанице. Этим же явлением объясняется относительная эффективность алгоритма R (хотя здесь предпочтение отдается анализу, начатому с младших разрядов), потому что никогда не приходится работать более чем с M стопками и стопки приходится сцеплять всего p раз. С другой стороны, нетрудно построить СЦ-поразрядный метод с помощью связного распределения памяти с отрицательными связями для обозначения границ между стопками, как в алгоритме 5.2.4L (см. упр. 10). Пожалуй, наилучший компромиссный выход указал М. Д. Мак-Ларен (M. D. MacLaren) [*JACM* **13** (1966), 404–411], который предложил использовать МЦ-сортировку, как в алгоритме R, но *в применении к старшим разрядам.* Это не будет полной сортировкой массива, но в результате массив станет почти упорядоченным, т. е. в нем останется очень мало инверсий, так что для завершения сортировки можно будет воспользоваться методом простых вставок. Наш анализ программы 5.2.1M применим и к этой ситуации; если ключи распределены равномерно, то после сортировки массива по старшим p цифрам в нем останется в среднем $\frac{1}{4}N(N-1)M^{-p}$ инверсий [см. формулу 5.2.1–(17) и упр. 5.2.1–38]. Мак-Ларен вычислил среднее число обращений к памяти, приходящихся на один обрабатываемый элемент, и оказалось, что оптимальный выбор значений M и p (в предположении, что M — степень двойки, ключи равномерно распределены и $N/M^p \le 0.1$, так что отклонения от равномерного распределения приемлемы) описывается следующей таблицей.

	$N =$	100	1000	10000	100000	1000000	10^7	10^8	10^9
Наилучшее	$M =$	32	128	512	1024	8192	2^{15}	2^{17}	2^{19}
Наилучшее	$p =$	2	2	2	2	2	2	2	2
	$\beta(N) =$	19.3	18.5	18.2	18.1	18.0	18.0	18.0	18.0

Здесь $\beta(N)$ — среднее число обращений к памяти на один сортируемый элемент

$$\beta(N) = 5p + 8 + \frac{2pM}{N} + \frac{N-1}{2M^p} - \frac{H_N}{N}; \tag{8}$$

эта величина ограничена при $N \to \infty$, если взять $p = 2$ и $M > \sqrt{N}$, так что среднее время сортировки — $O(N)$, а не $N \log N$. Данный метод является усовершенствованным вариантом метода вставок в несколько списков (программа 5.2.1M), который, по существу, представляет собой случай $p = 1$. В упр. 12 приводится интересная процедура Мак-Ларена для окончательной перекомпоновки записей после частичной сортировки массива с помощью списков.

Если воспользоваться методами из алгоритма 5.2D и упр. 5.2–13, то можно обойтись без полей связи; при этом в дополнение к памяти, занятой самими записями, потребуется всего $O(\sqrt{N})$ ячеек. Если исходные данные распределены равномерно, то среднее время сортировки пропорционально N.

В. Добосевич (W. Dobosiewicz) получил хороший результат при использовании СЦ-сортировки в отношении массивов значительной длины. Процесс распределения был построен таким образом, что для первых $M/2$ стопок гарантировалось получе-

ние от 25 до 75% всех записей [см. *Inf. Proc. Letters* **7** (1978), 1–6; **8** (1979), 170–172]. В результате среднее время сортировки равномерно распределенных ключей оказалось порядка $O(N)$, но для наихудшего случая время будет составлять порядка $O(N \log N)$. Указанная статья побудила других исследователей разработать новые алгоритмы вычисления адресов, из которых наиболее удачной оказалась следующая двухуровневая схема, предложенная Маркку Тамминеном (Markku Tamminen) [*J. Algorithms* **6** (1985), 138–144]. Предположим, что все ключи являются дробями из интервала $[0\,..\,1)$. Сначала распределим N записей по $\lfloor N/8 \rfloor$ хранилищам и поставим в соответствие ключу K хранилище $\lfloor KN/8 \rfloor$. Далее предположим, что в хранилище k оказалось N_k записей. Если $N_k \le 16$, можно использовать сортировку методом прямых вставок, в противном случае — сортировать так же, как предлагал Мак-Ларен, но для M^2 хранилищ (стопок), где $M^2 \approx 10N_k$. Тамминен доказал следующую весьма существенную теорему.

Теорема Т. *Существует константа T, такая, что описанный выше метод сортировки требует в среднем не более TN операций, если только ключи являются независимыми случайными числами с ограниченной и интегрируемой по Риману плотностью распределения $f(x)$ при $0 \le x \le 1$.* (Константа T не зависит от вида функции f.)

Доказательство. (См. упр. 18.) Интуитивно ясно, что в результате выполнения первой фазы распределения между $N/8$ стопками будет найден интервал, на котором функция f является почти постоянной; на второй фазе ожидаемый размер стопки станет почти постоянным. ▌

Некоторые версии поразрядной сортировки были доведены до совершенства при работе с большими массивами буквенных строк, что описано в весьма содержательной статье P. M. McIlroy, K. Bostic, M. D. McIlroy, *Computing Systems* **6** (1993), 5–27.

УПРАЖНЕНИЯ

▶ **1.** [*20*] Алгоритм из упр. 5.2–13 показывает, как можно выполнить распределяющую сортировку, имея объем памяти, достаточный для хранения всего N записей (и M полей счетчиков), а не $2N$ записей. Приводит ли эта идея к усовершенствованию алгоритма поразрядной сортировки, проиллюстрированного в табл. 1?

2. [*13*] Является ли алгоритм R алгоритмом устойчивой сортировки?

3. [*15*] Объясните, почему в алгоритме H выходит так, что `BOTM[0]` указывает на первую запись в "объединенной" очереди даже в случае, *если стопка 0 оказывается пустой.*

▶ **4.** [*23*] Во время выполнения алгоритма R все M стопок хранятся в виде связных очередей ("первым вошел — первым вышел"). Исследуйте идею связывания элементов стопок, как в стеке. (На рис. 33 стрелки будут указывать не вверх, а вниз и таблица `BOTM` станет не нужна.) Покажите, что если сцеплять стопки в соответствующем порядке, то может получиться работоспособный метод сортировки. Будет ли этот алгоритм более простым или более быстрым?

5. [*20*] Какие изменения необходимо внести в программу R, чтобы она выполняла сортировку не трехбайтовых ключей, а восьмибайтовых? Считается, что старшие байты ключа K_i хранятся по адресу `KEY` $+\, i(1{:}5)$, а три младших байта, как и раньше, — по адресу `INPUT` $+\, i(1{:}3)$. Каково время выполнения программы с этими изменениями?

6. [*M24*] Пусть $g_{MN}(z) := \sum p_{MNk} z^k$, где p_{MNk} — вероятность того, что после случайного просмотра поразрядной сортировки, разложившего N элементов на M стопок, получилось ровно k пустых стопок.

a) Покажите, что $g_{M(N+1)}(z) = g_{MN}(z) + ((1-z)/M)g'_{MN}(z)$.

b) Найдите при помощи указанного соотношения простые выражения для математического ожидания и дисперсии этого распределения вероятностей как функций от M и N.

7. [*20*] Проанализируйте, в чем состоит сходство и отличие алгоритма R и обменной поразрядной сортировки (алгоритм 5.2.2R).

▸ **8.** [*20*] В алгоритмах поразрядной сортировки, обсуждавшихся в тексте раздела, предполагалось, что все сортируемые ключи неотрицательны. Какие изменения следует внести в эти алгоритмы в том случае, когда ключами являются и отрицательные числа, представленные в *дополнительном* или *обратном* коде?

9. [*20*] (Продолжение упр. 8.) Какие изменения нужно внести в эти алгоритмы в случае, когда ключами являются числа, представленные в виде *абсолютной величины со знаком*?

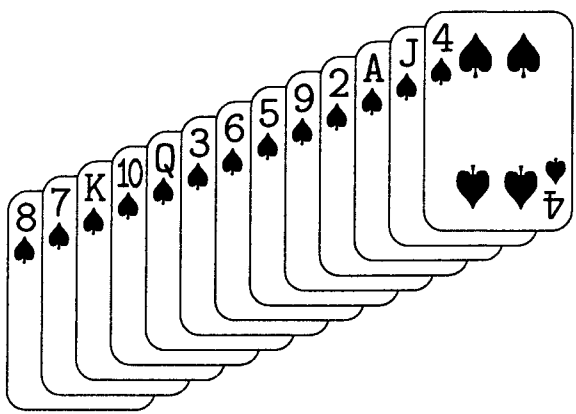
10. [*30*] Разработайте алгоритм поразрядной СЦ-сортировки, использующий связное распределение памяти. (Так как размер подмассивов все уменьшается, разумно уменьшить M, а для сортировки коротких массивов применить метод, отличный от поразрядной сортировки.)

11. [*16*] Перестановка 16 исходных чисел, показанная в табл. 1, содержит вначале 41 инверсию. По завершении сортировки инверсий, разумеется, нет совсем. Сколько инверсий осталось бы в массиве, если пропустить первый просмотр и выполнить поразрядную сортировку лишь по цифрам десятков и сотен? Сколько инверсий останется, если пропустить как первый, так и второй просмотры?

12. [*24*] (М. Д. Мак-Ларен (M. D. MacLaren).) Предположим, алгоритм R применили только к p старшим цифрам реальных ключей; тогда массив, если его читать по порядку, указанному связями, почти рассортирован, но ключи, старшие p цифр которых совпадают, могут быть неупорядочены. Придумайте такой алгоритм перекомпоновки записей в пределах того же объема памяти, чтобы ключи расположились в порядке $K_1 \le K_2 \le \cdots \le K_N$. [*Указание.* Частный случай, когда массив полностью рассортирован, можно найти в ответе к упр. 5.2–12; его можно скомбинировать с простыми вставками без потери эффективности, так как в массиве осталось мало инверсий.]

13. [*40*] Реализуйте метод внутренней сортировки, предложенный в тексте в конце этого раздела, и разработайте программу сортировки случайных данных за $O(N)$ машинных циклов, требующую всего $O(\sqrt{N})$ дополнительных ячеек памяти.

14. [*22*] Последовательность игральных карт можно рассортировать в порядке возрастания от верхней карты к нижней за два просмотра, раскладывая их каждый раз лишь в две стопки:



Разложите карты лицевой стороной вниз в две стопки, содержащие A 2 ... J Q K соответственно (от нижней карты к верхней). Затем положите вторую стопку поверх первой, поверните колоду лицевой стороной вверх и разложите в две стопки: A 2 9 3 10 и 4 J 5 6 Q K 7 8. Сложите эти две стопки и поверните их лицевой стороной вверх. Колода будет рассортирована.

Докажите, что приведенную выше последовательность карт нельзя рассортировать в порядке *убывания* K Q J ... 2 A от верхней карты к нижней за два просмотра, даже если разрешено раскладывать карты в три стопки. (Сдавать карты нужно всегда сверху колоды, поворачивая их рубашкой вверх. На рисунке верхняя карта колоды изображена справа, а нижняя — слева.)

15. [*M25*] Рассмотрите задачу из упр. 14 для случая, когда карты раздаются лицевой стороной вверх, а не вниз. Таким образом, один просмотр можно потратить на преобразование возрастающего порядка в убывающий. Сколько нужно просмотров?

▶ **16.** [*25*] Разработайте алгоритм сортировки строк $\alpha_1, \ldots, \alpha_n$, состоящих из m букв в лексикографическом порядке. Суммарное время выполнения этого алгоритма должно быть порядка $O(m+n+N)$, где $N = |\alpha_1| + \cdots + |\alpha_n|$ — суммарная длина всех строк.

17. [*15*] Почему в двухуровневом методе сортировки, предложенном Тамминеном (см. теорему T), метод, аналогичный алгоритму Мак-Ларена, используется на втором и не используется на первом уровне?

18. [*HM26*] Докажите теорему T. *Указание.* Сначала покажите, что алгоритм Мак-Ларена действительно выполняет в среднем $O(BN)$ операций, если его применять по отношению к независимым случайным ключам, для которых функция плотности вероятностей удовлетворяет условию $f(x) \le B$ при $0 \le x \le 1$.

Для сортировки корней и слов
нам понадобится 1 100 ящиков
и лотков для форм.

— ДЖОРДЖ В. ВИГРАМ (GEORGE V. WIGRAM) (1843)

5.3. ОПТИМАЛЬНАЯ СОРТИРОВКА

Теперь, когда мы проанализировали множество методов внутренней сортировки, пришло время обратиться к более общему вопросу: *какой метод внутренней сортировки наилучший?* Существует ли такой верхний предел скорости сортировки, которого бы не мог достичь ни один программист, как бы искусен он ни был?

Разумеется, *не существует* наилучшего возможного способа сортировки; мы должны точно определить, что понимать под словом "наилучший", но не существует наилучшего возможного способа определения слова "наилучший". Аналогичную проблему, связанную с оптимальностью алгоритмов, мы обсуждали в разделах 4.3.3, 4.6.3 и 4.6.4, в которых рассматривались умножение с повышенной точностью и вычисление полиномов. В каждом случае, для того чтобы задача была поставлена в терминах конкретных математических структур, необходимо было сформулировать довольно простое определение "наилучшего из возможных" алгоритма. И в каждом случае перед нами вставали интереснейшие проблемы, настолько сложные, что они до сих пор полностью не решены. Так же обстоит дело и с сортировкой: были получены некоторые интересные результаты, но осталось еще много интригующих вопросов, на которые до сих пор нет ответов.

Изучение внутреннего механизма методов сортировки обычно было направлено на минимизацию числа сравнений ключей при сортировке n элементов или слиянии m элементов с n элементами, или выборе t-го наибольшего элемента из неупорядоченного набора n элементов. В разделах 5.3.1–5.3.3 эти вопросы обсуждаются для общего случая; в разделе 5.3.4 рассматриваются аналогичные вопросы с интересным ограничением: схема (структура) сравнений должна быть, по сути, заранее фиксированной. Другие интересные теоретические вопросы, связанные с оптимальной сортировкой, можно найти в упражнениях к разделу 5.3.4 и в разделах 5.4.4, 5.4.8 и 5.4.9, в которых анализируется внешняя сортировка.

Как только появится аналитическая машина, она, безусловно, определит дальнейший путь развития науки. Всякий раз, когда с ее помощью будет найден какой-либо результат, возникнет вопрос: "Существует ли метод вычислений, которым можно получить на этой машине тот же результат, но затратив минимум времени?".

— ЧАРЛЬЗ БЭББИДЖ (1864)

5.3.1. Сортировка с минимальным числом сравнений

Очевидно, минимальное число сравнений ключей, необходимое для сортировки n элементов, равно *нулю*, поскольку, как мы видели, существуют методы поразрядной сортировки, в которых сравнения вообще не выполняются. В самом деле, можно разработать MIX-программы, способные сортировать и, тем не менее, не содержащие ни одной команды условного перехода! (См. упр. 5–8 в начале этой главы.) Мы также встречались с несколькими методами сортировки, которые, по сути, были основаны на сравнении ключей, но время выполнения которых на деле определялось другими факторами, такими как перезапись данных, вспомогательные операции и т. д.

Поэтому ясно, что подсчет числа сравнений — не единственный способ измерения эффективности метода сортировки. Однако в любом случае небезынтерес-

но провести тщательное исследование числа сравнений, поскольку теоретический анализ этого вопроса позволит нам более глубоко проникнуть во внутреннюю природу процессов сортировки, а также поможет отточить мастерство для решения задач, более близких к практике, которые могут возникнуть перед нами в будущем. Исключим из рассмотрения метод поразрядной сортировки, в котором совсем не выполняется сравнений, и ограничимся обсуждением методов, основанных только на абстрактном линейном отношении порядка "$<$" между ключами, рассмотренном в начале этой главы. Для простоты мы также ограничимся случаем *различных ключей*, а это значит, что при любом сравнении ключей K_i и K_j возможны лишь два исхода: либо $K_i < K_j$, либо $K_i > K_j$. (Распространение результатов такого анализа на общий случай, когда допускаются равные ключи, приводится в упр. 3–12.) Ограничения на время выполнения в худшем случае рассматриваются в работах Fredman, Willard, *J. Computer и Syst. Sci.* **47** (1993), 424–436, Ben-Amram, Galil, *J. Comp. Syst. Sci.* **54** (1997), 345–370 и Thorup, *SODA* **9** (1998), 550–555.

Возможны и другие эквивалентные варианты постановки задачи сортировки посредством сравнений. Если есть n грузов и весы с двумя чашами, то каково минимальное число взвешиваний, необходимое для того, чтобы расположить грузы по порядку в соответствии с весом, если в каждой чаше весов помещается только один груз? Или же, если в некотором турнире участвуют n игроков, то каково наименьшее число парных встреч, достаточное для того, чтобы распределить места между соревнующимися в предположении, что силы игроков можно линейно упорядочить (ничейные результаты не допускаются).

Методы сортировки n элементов, удовлетворяющие указанным ограничениям, можно представить с помощью структуры расширенного бинарного дерева, такого, которое показано на рис. 34. Каждый *внутренний узел* (изображенный в виде кружочка) содержит два индекса "$i:j$" и означает сравнение ключей K_i и K_j. Левое поддерево этого узла соответствует последующим сравнениям, которые необходимо выполнить, если $K_i < K_j$, а правое поддерево — тем действиям, которые необходимо предпринять, если $K_i > K_j$. Каждый *внешний узел дерева* (изображенный в виде прямоугольника) содержит перестановку $a_1\, a_2 \ldots a_n$ множества $\{1, 2, \ldots, n\}$, а это обозначает, что было установлено упорядочение

$$K_{a_1} < K_{a_2} < \cdots < K_{a_n}.$$

(Если взглянуть на путь от корня к этому внешнему узлу, то каждое из $n-1$ соотношений $K_{a_i} < K_{a_{i+1}}$, где $1 \le i < n$, — результат некоторого сравнения $a_i:a_{i+1}$ или $a_{i+1}:a_i$ на этом пути.)

Так, на рис. 34 представлен метод сортировки, суть которого состоит в том, чтобы сначала сравнить K_1 с K_2; если $K_1 > K_2$, то продолжать (двигаясь по правому поддереву) сравнивать K_2 с K_3, а затем, если $K_2 < K_3$, сравнить K_1 с K_3; наконец, если $K_1 > K_3$, становится ясно, что $K_2 < K_3 < K_1$. Реальный алгоритм сортировки будет также перезаписывать ключи в массиве, но нас интересуют только сравнения, поэтому мы игнорируем все перезаписи данных. При сравнении K_i с K_j в этом дереве всегда имеются в виду *исходные ключи* K_i и K_j, а не те ключи, которые могли занять i- и j-ю позиции в массиве в результате перемещения записей.

Возможны и избыточные сравнения; например, на рис. 35 нет необходимости сравнивать $3:1$, поскольку из неравенств $K_1 < K_2$ и $K_2 < K_3$ следует $K_1 < K_3$. Ни-

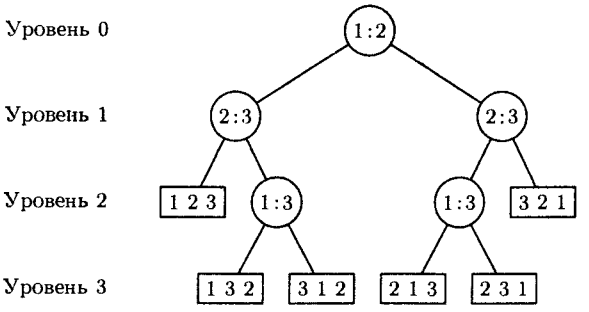


Рис. 34. Дерево сравнений для сортировки трех элементов.

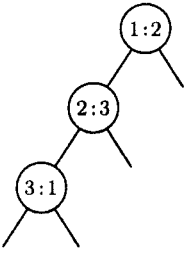


Рис. 35. Пример избыточного сравнения.

какая перестановка не может соответствовать левому поддереву узла 3:1 на рис. 35, так что эта часть алгоритма никогда не будет выполняться! Поскольку нас интересует минимальное число сравнений, можно считать, что избыточные сравнения не выполняются. С этого момента мы будем иметь дело со структурой расширенного бинарного дерева, в котором каждому внешнему узлу соответствует некоторая перестановка. Все перестановки исходных ключей возможны, и каждая перестановка определяет единственный путь от корня к внешнему узлу; отсюда вытекает, что *в дереве сравнений для сортировки n элементов без избыточных сравнений имеется ровно $n!$ внешних узлов.*

Оптимизация в наихудшем случае. Первая естественным образом возникающая задача — найти деревья сравнений, минимизирующие *максимальное* число выполняемых сравнений. (Позже мы рассмотрим *среднее* число сравнений.)

Пусть $S(n)$ — минимальное число сравнений, достаточное для сортировки n элементов. Если все внутренние узлы в дереве сравнений располагаются на уровнях $< k$, то очевидно, что в дереве не может быть более 2^k внешних узлов. Следовательно, полагая $k = S(n)$, имеем

$$n! \le 2^{S(n)}.$$

Поскольку $S(n)$ — целое число, можно записать эту формулу иначе, получив нижнюю оценку:

$$S(n) \ge \lceil \lg n! \rceil. \tag{1}$$

Заметим, что по формуле Стирлинга

$$\lceil \lg n! \rceil = n \lg n - n/\ln 2 + \tfrac{1}{2} \lg n + O(1), \tag{2}$$

следовательно, необходимо выполнить около $n \lg n$ сравнений. Соотношение (1) часто называют *теоретико-информационной нижней оценкой*, поскольку специалист в области теории информации сказал бы, что в процессе сортировки проверяется $\lg n!$ "бит информации"; каждое сравнение дает не более одного бита информации. Такие деревья, как на рис. 34, называют также "вопросниками" (questionnaires); их математические свойства исследованы в книге Claude Picard, *Théorie des Questionnaires* (Paris: Gauthier-Villars, 1965).

Из всех рассмотренных нами методов сортировки три метода требуют меньше всего сравнений: бинарные вставки (см. раздел 5.2.1), выбор из дерева (см. раздел 5.2.3) и простое двухпутевое слияние, описанное в алгоритме 5.2.4L). Нетрудно видеть, что максимальное число сравнений для метода бинарных вставок равно

$$B(n) = \sum_{k=1}^{n} \lceil \lg k \rceil = n\lceil \lg n \rceil - 2^{\lceil \lg n \rceil} + 1 \tag{3}$$

(см. упр. 1.2.4–42), а максимальное число сравнений для двухпутевого слияния определено в упр. 5.2.4–14. Оказывается (см. раздел 5.3.3), что для метода выбора из дерева верхняя оценка числа сравнений либо такая же, как для метода бинарных вставок, либо такая же, как для двухпутевого слияния, в зависимости от вида дерева. Во всех трех случаях имеем асимптотическое значение $n \lg n$; объединяя верхнюю и нижнюю оценки для $S(n)$, получим

$$\lim_{n\to\infty} \frac{S(n)}{n \lg n} = 1. \tag{4}$$

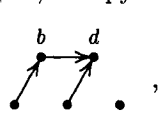
Таким образом, мы получили приближенную формулу для $S(n)$, однако желательно иметь более точную оценку. В следующей таблице приведены значения верхней и нижней границ при малых n.

$n =$	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
$\lceil \lg n! \rceil =$	0	1	3	5	7	10	13	16	19	22	26	29	33	37	41	45	49
$B(n) =$	0	1	3	5	8	11	14	17	21	25	29	33	37	41	45	49	54
$L(n) =$	0	1	3	5	9	11	14	17	25	27	30	33	38	41	45	49	65

Здесь $B(n)$ и $L(n)$ относятся соответственно к методам бинарных вставок и слияния списков. Можно показать, что $B(n) \le L(n)$ при любом n (см. упр. 2).

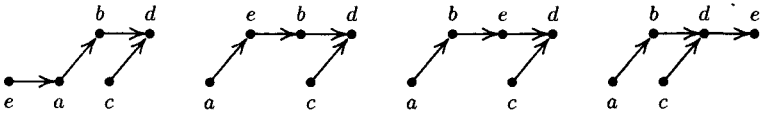
Как видно из приведенной выше таблицы, $S(4) = 5$, но $S(5)$ может равняться либо 7, либо 8. В результате снова приходим к задаче, поставленной в начале раздела 5.2. Каков наилучший способ сортировки пяти элементов? Возможна ли сортировка пяти элементов при помощи всего семи сравнений?

Такая сортировка возможна, но сам способ найти не так просто. Начинаем так же, как при сортировке четырех элементов посредством слияний, сравнивая сначала $K_1 : K_2$, затем — $K_3 : K_4$, а затем — наибольшие элементы обеих пар. Эти сравнения порождают конфигурацию, которую можно изобразить диаграммой



, (5)

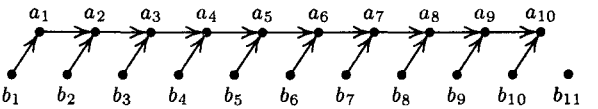
показывающей, что $a < b < d$ и $c < d$. (Для представления известных отношений порядка между элементами удобно воспользоваться ориентированными графами, такими, что x считается меньше y тогда и только тогда, когда на графе есть путь от x к y.) Теперь вставляем пятый элемент $K_5 = e$ в соответствующее место среди $\{a, b, d\}$; для этого требуются всего два сравнения, поскольку можно сравнить его сначала с b, а затем — с a или d. Таким образом, остается один из четырех возможных вариантов



(6)

и в каждом случае достаточно еще двух сравнений, чтобы вставить c в цепочку остальных элементов, меньших d. Такой способ сортировки пяти элементов впервые обнаружил Г. Б. Демут (H. B. Demuth) [Ph. D. thesis, Stanford University (1956), 41–43].

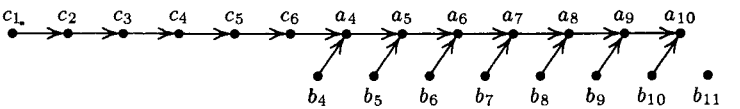
Сортировка посредством вставок и слияния. Изящное обобщение изложенного выше метода принадлежит Лестеру Форду (мл.) (Lester Ford, Jr.) и Селмеру М. Джонсону (Selmer M. Johnson). Поскольку оно объединяет некоторые особенности двух способов сортировки (посредством слияний и посредством вставок), мы назовем этот метод *сортировкой посредством вставок и слияния*. Рассмотрим, например, сортировку 21 элемента. Начать можно со сравнений десяти пар ключей $K_1 : K_2, K_3 : K_4, \ldots, K_{19} : K_{20}$; затем следует рассортировать посредством вставок и слияния бóльшие элементы пар. В результате получим конфигурацию



,

(7)

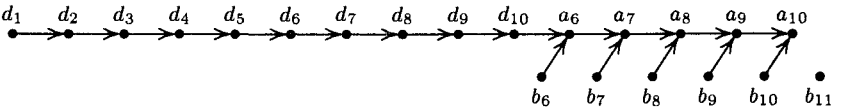
аналогичную (5). Следующий шаг — вставить элемент b_3 в последовательность $\{b_1, a_1, a_2\}$, а затем — b_2 в последовательность остальных элементов, меньших a_2. В результате приходим к конфигурации



.

(8)

Назовем верхние элементы *главной цепочкой*. Элемент b_5 можно вставить в главную цепочку за три сравнения (сравнив его сначала с c_4, затем с c_2 или c_6 и т. д.). После этого еще за три сравнения можно переместить в главную цепочку b_4 и получить



.

(9)

"Следующий шаг решающий; ясно ли вам, что делать дальше?" При помощи всего четырех сравнений вставляем b_{11} (но *не* b_7) в главную цепочку. Далее элементы b_{10}, b_9, b_8, b_7, b_6 (именно в таком порядке) можно вставить в нужное место в главной цепочке не более чем за четыре сравнения каждый.

Аккуратный подсчет числа требуемых сравнений показывает, что 21 элемент можно рассортировать не более чем за $10+S(10)+2+2+3+3+4+4+4+4+4+4 = 66$ шагов. Поскольку

$$2^{65} < 21! < 2^{66},$$

ясно также, что и в любом другом случае необходимо не менее 66 сравнений; следовательно,

$$S(21) = 66. \tag{10}$$

(При сортировке с помощью бинарных вставок понадобилось бы 74 сравнения.)

В общем случае сортировка посредством вставок и слияния для n элементов выглядит следующим образом.

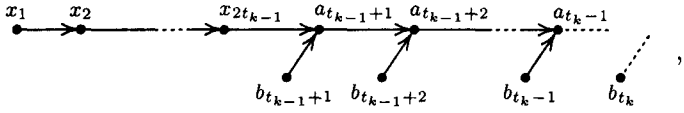
i) Сравнить $\lfloor n/2 \rfloor$ непересекающихся пар элементов. (Если n нечетно, то один элемент не участвует в сравнениях.)

ii) Рассортировать $\lfloor n/2 \rfloor$ бóльших элементов пар, найденных на шаге (i), посредством вставок и слияния.

iii) Для элементов введем обозначения $a_1, a_2, \ldots, a_{\lfloor n/2 \rfloor}$, $b_1, b_2, \ldots, b_{\lceil n/2 \rceil}$, как в (7), где $a_1 \le a_2 \le \cdots \le a_{\lfloor n/2 \rfloor}$ и $b_i \le a_i$ при $1 \le i \le \lfloor n/2 \rfloor$; назовем b_1 и все элементы a главной цепочкой. Не трогая элементов b_j при $j > \lceil n/2 \rceil$, вставим посредством бинарных вставок в главную цепочку остальные элементы b в следующем порядке:

$$b_3, b_2;\ b_5, b_4;\ b_{11}, b_{10}, \ldots, b_6;\ \ldots;\ b_{t_k}, b_{t_k-1}, \ldots, b_{t_{k-1}+1};\ \ldots. \tag{11}$$

Наша цель — сформировать последовательность $(t_1, t_2, t_3, t_4, \ldots) = (1, 3, 5, 11, \ldots)$, присутствующую в (11), таким образом, чтобы каждый из элементов $b_{t_k}, b_{t_k-1}, \ldots, b_{t_{k-1}+1}$ можно было вставить в главную цепочку не более чем за k сравнений. Обобщая (7)–(9), получим диаграмму



,

на которой главная цепочка до a_{t_k-1} включительно содержит $2t_{k-1} + (t_k - t_{k-1} - 1)$ элементов. Это число должно быть меньше 2^k; для нас лучше всего положить его равным $2^k - 1$, и тогда

$$t_{k-1} + t_k = 2^k. \tag{12}$$

Поскольку $t_1 = 1$, для удобства можно положить $t_0 = 1$. В итоге, суммируя члены геометрической прогрессии, найдем

$$\begin{aligned} t_k = 2^k - t_{k-1} = 2^k - 2^{k-1} + t_{k-2} = \cdots &= 2^k - 2^{k-1} + \cdots + (-1)^k 2^0 \\ &= \bigl(2^{k+1} + (-1)^k\bigr)/3. \end{aligned} \tag{13}$$

(Любопытно, что точно такая же последовательность возникла при изучении алгоритма вычисления наибольшего общего делителя двух целых чисел; см. упр. 4.5.2–36.)

Пусть $F(n)$ — число сравнений, необходимых для сортировки n элементов посредством вставок и слияния. Ясно, что

$$F(n) = \lfloor n/2 \rfloor + F(\lfloor n/2 \rfloor) + G\bigl(\lceil n/2 \rceil\bigr), \tag{14}$$

где функция G описывает объем работы, выполняемой на шаге (iii). Если $t_{k-1} \le m \le t_k$, то, суммируя по частям, получаем

$$G(m) = \sum_{j=1}^{k-1} j(t_j - t_{j-1}) + k(m - t_{k-1}) = km - (t_0 + t_1 + \cdots + t_{k-1}). \tag{15}$$

Положим

$$w_k = t_0 + t_1 + \cdots + t_{k-1} = \lfloor 2^{k+1}/3 \rfloor, \tag{16}$$

и тогда $(w_0, w_1, w_2, w_3, w_4, \ldots) = (0, 1, 2, 5, 10, 21, \ldots)$. В упр. 13 показано, что

$$F(n) - F(n-1) = k \qquad \text{тогда и только тогда, когда} \qquad w_k < n \le w_{k+1}, \tag{17}$$

а последнее условие эквивалентно неравенствам

$$\frac{2^{k+1}}{3} < n \le \frac{2^{k+2}}{3}$$

или $k + 1 < \lg 3n \le k + 2$; следовательно,

$$F(n) - F(n-1) = \lceil \lg \tfrac{3}{4} n \rceil. \tag{18}$$

(Эта формула получена А. Хадьяном (A. Hadian) [Ph. D. thesis, Univ. of Minnesota (1969), 38–42].) Отсюда вытекает, что функция $F(n)$ выражается на удивление простой формулой

$$F(n) = \sum_{k=1}^{n} \lceil \lg \tfrac{3}{4} k \rceil, \tag{19}$$

которая очень похожа на формулу (3) для метода бинарных вставок. В замкнутом виде эту сумму можно найти в упр. 14.

Воспользовавшись (19), нетрудно построить таблицу значений функции $F(n)$; имеем

$n =$	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
$\lceil \lg n! \rceil =$	0	1	3	5	7	10	13	16	19	22	26	29	33	37	41	45	49
$F(n) =$	0	1	3	5	7	10	13	16	19	22	26	30	34	38	42	46	50

$n =$	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33
$\lceil \lg n! \rceil =$	53	57	62	66	70	75	80	84	89	94	98	103	108	113	118	123
$F(n) =$	54	58	62	66	71	76	81	86	91	96	101	106	111	116	121	126

Обратите внимание на то, что $F(n) = \lceil \lg n! \rceil$ при $1 \le n \le 11$ и при $20 \le n \le 21$; таким образом, при этих значениях n сортировка посредством вставок и слияния оптимальна:

$$S(n) = \lceil \lg n! \rceil = F(n) \qquad \text{при } n = 1, \ldots, 11, 20 \text{ и } 21. \tag{20}$$

Задачу нахождения функции $S(n)$ поставил Гуго Штейнгауз (Hugo Steinhaus) во втором издании своей классической книги *Mathematical Snapshots* (Oxford University Press, 1950, 38–39)*. Он описал метод бинарных вставок, который является наилучшим способом сортировки n элементов при условии, что n-й элемент не рассматривается до тех пор, пока не рассортированы первые $n - 1$ элементов; он

* Есть русский перевод первого издания этой книги: Штейнгауз Г. Математический калейдоскоп. — М.: Гостехиздат, 1949. — *Прим. перев.*

Таблица 1

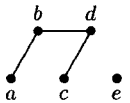
ЗНАЧЕНИЯ ФАКТОРИАЛОВ В ДВОИЧНОЙ СИСТЕМЕ СЧИСЛЕНИЯ

$$(1)_2 = 1!$$
$$(10)_2 = 2!$$
$$(110)_2 = 3!$$
$$(11000)_2 = 4!$$
$$(1111000)_2 = 5!$$
$$(1011010000)_2 = 6!$$
$$(1001110110000)_2 = 7!$$
$$(1001110110000000)_2 = 8!$$
$$(1011000100110000000)_2 = 9!$$
$$(110111010111100000000)_2 = 10!$$
$$(1001100001000101010000000000)_2 = 11!$$
$$(111001000110011111100000000000)_2 = 12!$$
$$(1011100110010100011001100000000000)_2 = 13!$$
$$(10100010011000011101100101000000000000)_2 = 14!$$
$$(100110000011101110111011101011000000000000)_2 = 15!$$
$$(1001100000111011101110111010110000000000000000)_2 = 16!$$
$$(10100001101111110111011101100110110000000000000000)_2 = 17!$$
$$(101101011111011101100110010100111001100000000000000000)_2 = 18!$$
$$(1101100000010101110010011000001101000100100000000000000000)_2 = 19!$$
$$(100001110000110110011101111100100000101011010000000000000000000)_2 = 20!$$

же сделал предположение о том, что метод бинарных вставок оптимален и в общем случае. Несколько лет спустя Штейнгауз сообщил [*Calcutta Math. Soc. Golden Jubilee Commemoration* **2** (1959), 323–327], что двое его коллег, С. Трибула (S. Trybuła) и Чжен Пинг (P. Czen), "недавно" опровергли его предположение и нашли значения $S(n)$ при $n \leq 11$. Вероятно, Трибула и Чжен Пинг независимо пришли к сортировке посредством вставок и слияния, о чем вскоре появилась публикация Ford, Johnson, *AMM* **66** (1959), 387–389.

После открытия сортировки посредством вставок и слияния первым неизвестным значением функции $S(n)$ оставалось $S(12)$. Из табл. 1 видно, что число 12! довольно близко к 2^{29}, поэтому существование 29-шаговой процедуры сортировки 12 элементов весьма маловероятно. Для решения этого вопроса Марком Уэлсом (Mark Wells) был предпринят продолжительный вычислительный эксперимент (продолжавшийся на компьютере Maniac II около 60 ч), который показал, что $S(12) = 30$ [*Proc. IFIP Congress 65* **2** (1965), 497–498]. Итак, процедура вставок и слияний оказывается оптимальной и при $n = 12$.

***Более глубокий анализ.** Чтобы более тщательно исследовать функцию $S(n)$, внимательно изучим диаграммы частичного упорядочения, подобные (5). Полученные после нескольких сравнений сведения можно представить в виде ориентированного графа. Этот граф в силу транзитивности отношения "<" не содержит циклов. Следовательно, его всегда можно изобразить таким образом, чтобы все дуги были ориентированы слева направо; поэтому удобнее удалить из диаграммы стрелки. В результате диаграмма (5) преобразуется в



(21)

Пусть G — такой ориентированный граф; обозначим через $T(G)$ число перестановок, согласующихся с G, т. е. число способов разметки вершин графа G целыми числами $\{1, 2, \ldots, n\}$ так, чтобы число в вершине x было меньше числа в вершине y, если дуга $x \to y$ принадлежит G. Вот пример перестановки, согласующейся с (21): $a = 1$, $b = 4$, $c = 2$, $d = 5$, $e = 3$. Мы проанализировали функцию $T(G)$ для различных G в разделе 5.1.4, в котором было отмечено, что $T(G)$ есть число вариантов топологической сортировки графа G.

Пусть G — граф из n элементов, который можно получить после k сравнений; определим *эффективность* графа G функцией

$$E(G) = \frac{n!}{2^k T(G)}. \tag{22}$$

(Эта идея принадлежит Фрэнку Хвангу (Frank Hwang) и Шень Линю (Shen Lin).) Строго говоря, эффективность не есть функция лишь самого графа G — она зависит от того пути, которым мы пришли к G в процессе сортировки, однако для простоты закроем глаза на эту маленькую неточность. Выполнив еще одно сравнение элементов i и j, получим два графа (G_1 и G_2): один — для случая $K_i < K_j$, а другой — для случая $K_i > K_j$. Ясно, что

$$T(G) = T(G_1) + T(G_2).$$

Если $T(G_1) \geq T(G_2)$, то имеем

$$T(G) \leq 2T(G_1),$$

$$E(G_1) = \frac{n!}{2^{k+1}T(G_1)} = \frac{E(G)T(G)}{2T(G_1)} \leq E(G). \tag{23}$$

Следовательно, каждое сравнение приводит к графу меньшей или равной эффективности; нельзя увеличить эффективность за счет дополнительных сравнений.

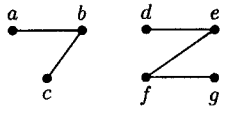
Заметим, что если G совсем не содержит дуг, то $k = 0$ и $T(G) = n!$, т. е. начальная эффективность равна 1. Если же граф G представляет окончательный результат сортировки, то он выглядит, как отрезок прямой, и $T(G) = 1$. Так, например, если нам нужно построить процедуру сортировки пяти элементов за семь или менее сравнений, то необходимо получить линейный граф •—•—•—•—•, эффективность которого равна $5!/(2^7 \times 1) = 120/128 = 15/16$. Отсюда следует, что все графы, возникающие в процессе сортировки, должны иметь эффективность $\geq \frac{15}{16}$; если бы появился какой-нибудь граф меньшей эффективности, то, по крайней мере, один из его потомков тоже имел бы меньшую эффективность и мы бы, в конце концов, пришли к линейному графу с эффективностью $< \frac{15}{16}$. В общем случае это рассуждение показывает, что все графы, соответствующие узлам дерева для некоторой процедуры сортировки n элементов, должны иметь эффективность $\geq n!/2^l$, где l — число уровней в дереве (не считая внешних узлов). Это еще один способ доказательства неравенства $S(n) \geq \lceil \lg n! \rceil$, хотя такое рассуждение на самом деле не сильно отличается от приведенного выше.

Граф (21) имеет эффективность 1, поскольку $T(G) = 15$ и граф G был получен за три сравнения. Чтобы выяснить, какие вершины должны участвовать в следующем сравнении, можно построить *матрицу сравнений*

$$C(G) = \begin{array}{c} \\ a \\ b \\ c \\ d \\ e \end{array} \begin{array}{c} \begin{array}{ccccc} a & b & c & d & e \end{array} \\ \begin{pmatrix} 0 & 15 & 10 & 15 & 11 \\ 0 & 0 & 5 & 15 & 7 \\ 5 & 10 & 0 & 15 & 9 \\ 0 & 0 & 0 & 0 & 3 \\ 4 & 8 & 6 & 12 & 0 \end{pmatrix} \end{array}, \tag{24}$$

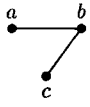
где C_{ij} есть $T(G_1)$ для графа G_1 T (Gi), полученного путем добавления дуги $i \to j$ в G. Если мы, например, сравним K_c с K_e, то 15 перестановок, согласующихся с G, распадутся на две группы: $C_{ec} = 6$, в которых $K_e < K_c$, и $C_{ce} = 9$, в которых $K_c < K_e$. Последний граф имел бы эффективность $15/(2 \times 9) = \frac{5}{6} < \frac{15}{16}$, так что это сравнение не может привести к процедуре сортировки из семи шагов. Если мы хотим сохранить эффективность $\geq \frac{15}{16}$, то следующим сравнением *обязано* быть $K_b : K_e$.

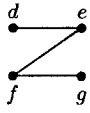
Концепция эффективности особенно полезна при рассмотрении связных компонентов графов. Возьмем, например, граф

$G =$ 

;

он состоит из двух компонентов:

$G' =$ 

и $G'' =$ 

.

В этих компонентах ни одна дуга не соединяет G' с G''; следовательно, он был образован путем нескольких сравнений вершин только G' и независимо нескольких сравнений вершин только G''. В общем случае предположим, что граф $G = G' \oplus G''$ не содержит дуг, связывающих G' и G'', где G' и G'' имеют соответственно n' и n'' вершин; легко видеть, что

$$T(G) = \binom{n' + n''}{n'} T(G')T(G''), \tag{25}$$

поскольку каждая перестановка, согласующаяся с G, получается в результате выбора n' элементов, которые считаются принадлежащими графу G', и последующего составления независимых перестановок, согласующихся с G' и G''. Пусть внутри G' выполнено k' сравнений, а внутри G'' — соответственно k'' сравнений; получаем основной результат

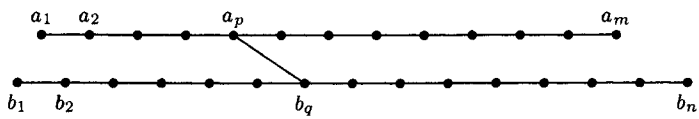
$$E(G) = \frac{(n' + n'')!}{2^{k'+k''}T(G)} = \frac{n'!}{2^{k'}T(G')} \frac{n''!}{2^{k''}T(G')} = E(G')E(G''), \tag{26}$$

показывающий, что между эффективностью графа и эффективностью его компонентов существует простая связь. Поэтому мы можем ограничить наше рассмотрение графами, имеющими только один компонент.

Теперь предположим, что G' и G'' — однокомпонентные графы и мы хотим связать их вместе, сравнив вершину x графа G' с вершиной y графа G''. Нужно выяснить, насколько эффективным получится новый граф. Для этого нам понадобится функция, которую можно обозначить через

$$\binom{p}{m} < \binom{q}{n}, \tag{27}$$

равная по определению числу перестановок, согласующихся с графом



(28)

Таким образом, $\binom{p}{m} < \binom{q}{n}$ есть произведение $\binom{m+n}{m}$ и вероятности того, что p-й наименьший элемент из множества m чисел меньше q-го наименьшего элемента из независимо выбранного множества n чисел. В упр. 17 показано, что функцию $\binom{p}{m} < \binom{q}{n}$ можно выразить через биномиальные коэффициенты двумя способами:

$$\begin{aligned}\binom{p}{m} < \binom{q}{n} &= \sum_{0\le k<q} \binom{m-p+n-k}{m-p}\binom{p-1+k}{p-1} \\ &= \sum_{p\le j\le m} \binom{n-q+m-j}{n-q}\binom{q-1+j}{q-1}.\end{aligned} \tag{29}$$

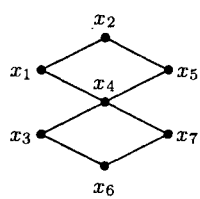
(Между прочим, алгебраически никоим образом не очевидно, что эти две суммы произведений биномиальных коэффициентов должны быть равны.) Справедливы такие равенства:

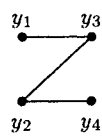
$$\binom{p}{m} < \binom{q}{n} + \binom{q}{n} < \binom{p}{m} = \binom{m+n}{m}; \tag{30}$$

$$\binom{q}{n} < \binom{p}{m} = \binom{m+1-p}{m} < \binom{n+1-q}{n}; \tag{31}$$

$$\binom{p}{m} < \binom{q}{n} = \binom{p}{m-1} < \binom{q}{n} + \binom{p}{m} < \binom{q}{n-1} + [p\le m][q=n]\binom{m+n-1}{m}. \tag{32}$$

Рассмотрим два графа:

$G' =$ 

, $G'' =$ 

. (33)

Нетрудно показать с помощью простого перечисления, что $T(G') = 42$ и $T(G'') = 5$; так что если G — граф с 11 вершинами, содержащий G' и G'' в качестве своих компонентов, то по формуле (25) $T(G) = \binom{11}{4} \cdot 42 \cdot 5 = 69300$. Такое число перестановок слишком внушительно, чтобы их можно было выписать и таким образом выяснить, в скольких из них $x_i < y_j$ для всех i и j. Однако это вычисление менее чем за час

можно проделать вручную следующим образом. Построим матрицы $A(G')$ и $A(G'')$, где A_{ik} — число перестановок, согласующихся с G' (или G''), в которых x_i (или y_i) равно k. Тогда число перестановок, согласующихся с G, в которых x_i меньше y_j, есть сумма по всем p и q ($1 \le p \le 7$ и $1 \le q \le 4$) произведений (i,p)-го элемента матрицы $A(G')$, $\binom{p}{7}<\binom{q}{4}$ и (j,q)-го элемента матрицы $A(G'')$. Иначе говоря, нужно вычислить произведение матриц $A(G') \cdot L \cdot A(G'')^T$, где $L_{pq} = \binom{p}{7}<\binom{q}{4}$. Оно равно

$$\begin{pmatrix} 21 & 16 & 5 & 0 & 0 & 0 & 0 \\ 0 & 5 & 10 & 12 & 10 & 5 & 0 \\ 21 & 16 & 5 & 0 & 0 & 0 & 0 \\ 0 & 0 & 12 & 18 & 12 & 0 & 0 \\ 0 & 0 & 0 & 0 & 5 & 16 & 21 \\ 0 & 5 & 10 & 12 & 10 & 5 & 0 \\ 0 & 0 & 0 & 0 & 5 & 16 & 21 \end{pmatrix} \begin{pmatrix} 210 & 294 & 322 & 329 \\ 126 & 238 & 301 & 325 \\ 70 & 175 & 265 & 315 \\ 35 & 115 & 215 & 295 \\ 15 & 65 & 155 & 260 \\ 5 & 29 & 92 & 204 \\ 1 & 8 & 36 & 120 \end{pmatrix} \begin{pmatrix} 2 & 3 & 0 & 0 \\ 2 & 2 & 0 & 1 \\ 1 & 0 & 2 & 2 \\ 0 & 0 & 3 & 2 \end{pmatrix} = \begin{pmatrix} 48169 & 42042 & 66858 & 64031 \\ 22825 & 16005 & 53295 & 46475 \\ 48169 & 42042 & 66858 & 64031 \\ 22110 & 14850 & 54450 & 47190 \\ 5269 & 2442 & 27258 & 21131 \\ 22825 & 16005 & 53295 & 46475 \\ 5269 & 2442 & 27258 & 21131 \end{pmatrix}.$$

Таким образом, "наилучший" способ соединить G' и G'' — сравнить x_1 с y_2; в 42042 случаях получим $x_1 < y_2$ и в $69300 - 42042 = 27258$ случаях — $x_1 > y_2$. (В силу симметрии, по существу, к тем же результатам привели бы сравнения x_3 с y_2, x_5 с y_3 и x_7 с y_3.) Эффективность полученного графа для $x_1 < y_2$ равна

$$\frac{69300}{84084} E(G')E(G''),$$

т. е. она не особенно высока; следовательно, по-видимому, вообще не стоит ни в одном методе сортировки применять соединение G' с G''! Смысл этого примера — показать, что мы смогли принять такое решение, не производя непомерно больших вычислений.

Этими идеями можно воспользоваться также для независимого подтверждения того, что $S(12) = 30$ (доказательство данного факта принадлежит Марку Уэлсу). Начав с графа, содержащего одну вершину, мы можем повторять попытки добавления сравнений к одному из наших графов G или к $G' \oplus G''$ (паре компонентов G' и G'') с таким расчетом, чтобы оба полученных графа содержали 12 или менее вершин и обладали эффективностью $\ge 12!/2^{29} \approx 0.89221$. Всякий раз, когда это оказывается возможным, мы выбираем граф с меньшей эффективностью и добавляем его к нашему множеству, если только он не является изоморфным одному из уже включенных в множество графов. Если оба полученных графа имеют одинаковую эффективность, то произвольным образом выбирается один из них. Граф можно приравнять к двойственному ему графу (т. е. полученному в результате обращения отношения порядка) при рассмотрении вопроса о добавлении сравнений как к $G' \oplus \mathrm{dual}(G'')$, так и к $G' \oplus G''$. Несколько графов, полученных таким способом, изображены на рис. 36, на котором приведены также значения их эффективности.

Прежде чем этот процесс завершился, при помощи компьютера было построено ровно 1 649 графов. Поскольку граф не был получен, можно сделать вывод о том, что $S(12) > 29$. Весьма правдоподобно, что и для доказательства неравенства $S(22) > 70$ можно выполнить аналогичный эксперимент за вполне разумное время, поскольку $22!/2^{70} \approx 0.952$ — это чрезвычайно высокая эффективность для сортировки за 70 шагов. (Из 1 649 найденных графов с 12 или менее вершинами всего 91 имеет столь высокую эффективность.)

Промежуточные результаты дают веские основания предположить, что $S(13) = 33$ и, следовательно, сортировка посредством вставок и слияния не оптимальна при

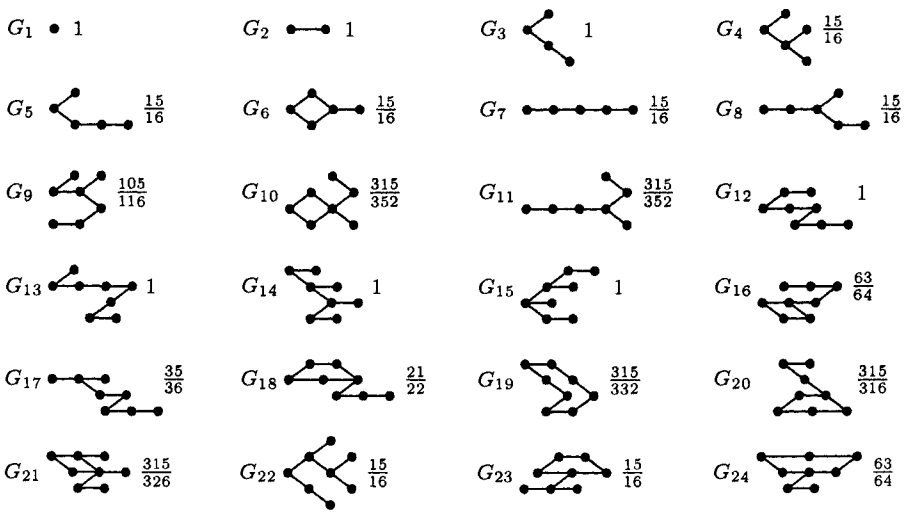


Рис. 36. Некоторые графы и значения их эффективности, полученные на начальной стадии длинного доказательства неравенства $S(12) > 29$.

$n = 13$. Наверняка можно доказать, что $S(16) < F(16)$, поскольку $F(16)$ — это как раз такое число сравнений, какое требуется, чтобы сначала рассортировать 10 элементов за $S(10)$ шагов, а затем посредством бинарных вставок вставить по одному остальные 6 элементов. Непременно должен существовать более хороший способ! Но в настоящее время вариант с наименьшим значением, в котором точно известно, что $F(n)$ неоптимально, — это $n = 47$: после сортировки 5 и 42 элементов, где требуется $F(5) + F(42) = 178$ сравнений, мы можем слить результаты, на что потребуется еще 22 сравнения, используя метод, предложенный в работе J. Schulte Mönting, *Theoretical Comp. Sci.* **14** (1981), 19–37; этот результат превышает значение $F(47) = 201$. (В работе Glenn K. Manacher, *JACM* **26** (1979), 441–456, показано, что существует бесконечно много значений n, начиная с $n = 189$, для которых $S(n) < F(n)$.)

Среднее число сравнений. До сих пор мы рассматривали процедуры, наилучшие в том смысле, что они не плохи в наихудшем случае; мы искали "минимаксные" процедуры, минимизирующие *максимальное* число сравнений. Поищем теперь "минисредние" процедуры, минимизирующие *среднее* число сравнений в предположении, что входные данные случайны, т. е. все перестановки равновероятны.

Рассмотрим еще раз изображенное на рис. 34 представление процедуры сортировки в виде дерева. Среднее число сравнений по всем перестановкам для этого дерева равно

$$\frac{2+3+3+3+3+2}{6} = 2\tfrac{2}{3}.$$

В общем случае среднее число сравнений для метода сортировки есть *длина внешнего пути дерева*, деленная на $n!$. (Напомним, что длина внешнего пути — это сумма всех расстояний от корня до каждого из внешних узлов; см. раздел 2.3.4.5.) Из анализа в разделе 2.3.4.5 следует, что минимальная длина внешнего пути достигается на таком бинарном дереве с N внешними узлами, у которого имеется $2^q - N$

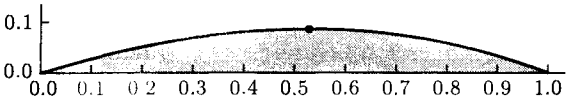


Рис. 37. Функция $1 + \theta - 2^\theta$.

внешних узлов на уровне $q - 1$ и $2N - 2^q$ на уровне q, где $q = \lceil \lg N \rceil$. (Корень находится на нулевом уровне.) Таким образом, минимальная длина внешнего пути равна

$$(q-1)(2^q - N) + q(2N - 2^q) = (q+1)N - 2^q. \tag{34}$$

Имеется еще один интересный способ охарактеризовать минимальную длину пути: *расширенное бинарное дерево имеет минимальную длину внешнего пути тогда и только тогда, когда существует такое число l,* что все *внешние узлы находятся на уровнях l и $l + 1$* (см. упр. 20).

Если положить $q = \lg N + \theta$, где $0 \le \theta < 1$, то формула минимальной длины внешнего пути примет вид

$$N(\lg N + 1 + \theta - 2^\theta). \tag{35}$$

График функции $1 + \theta - 2^\theta$ изображен на рис. 37; при $0 < \theta < 1$ она принимает положительные, но очень малые значения, не превышающие

$$1 - (1 + \ln\ln 2)/\ln 2 = 0.08607\ 13320\ 55934+. \tag{36}$$

Таким образом, минимальная возможная средняя длина пути, которая получается в результате деления (35) на N, не может быть меньше $\lg N$ и больше $\lg N + 0.0861$. (Этот результат впервые получил Э. Глисон (A. Gleason) в неопубликованной заметке 1956 года.)

Если теперь положим $N = n!$, то получим нижнюю оценку среднего числа сравнений по всем схемам сортировки. Оценка асимптотически приближается к

$$\lg n! + O(1) = n \lg n - n/\ln 2 + O(\log n). \tag{37}$$

Пусть $\bar{F}(n)$ — среднее число сравнений, выполняемых алгоритмом сортировки посредством вставок и слияния; имеем

$n =$	1	2	3	4	5	6	7	8
Нижняя граница (34) =	0	2	16	112	832	6896	62368	619904
$n!\,\bar{F}(n) =$	0	2	16	112	832	6912	62784	623232

Итак, сортировка посредством вставок и слияния оптимальна в обоих смыслах при $n \le 5$, однако при $n = 6$ такой метод в среднем требует $6912/720 = 9.6$ сравнений вместо возможных согласно нижней оценке $6896/720 = 9.577777\ldots$ сравнений. Немного поразмыслив, нетрудно понять, почему это так: некоторые "удачные" перестановки шести элементов сортируются методом вставок и слияний всего за восемь сравнений, и тогда дерево сравнений имеет внешние узлы на трех, а не на двух уровнях. Из-за этого увеличивается суммарная длина пути. В упр. 24 показано, что можно построить процедуру сортировки шести элементов, требующую всегда девять или десять сравнений; следовательно, этот метод превосходит метод вставок и слияний в среднем и не хуже него в худшем случае.

В работе Y. Césari, Thesis (Univ. of Paris, 1968), page 37, показано, что при $n = 7$ не существует метода сортировки, при котором достигалась бы нижняя граница длины внешнего пути (62 368). (Используя результат упр. 22, этот факт можно доказать самостоятельно.) С другой стороны, в указанной работе построены процедуры, для которых достигается нижняя граница (34) при $n = 9$ или 10. Вообще же, задачу минимизации среднего числа сравнений решить гораздо сложнее, чем найти $S(n)$. Вполне даже возможно, что при некоторых n все методы, минимизирующие среднее число сравнений, в худшем случае требуют *более* $S(n)$ сравнений.

УПРАЖНЕНИЯ

1. [*20*] Нарисуйте деревья сравнений для сортировки четырех элементов методами (a) бинарных вставок ; (b) простого двухпутевого слияния. Каковы длины внешних путей для этих деревьев?

2. [*M24*] Докажите, что $B(n) \le L(n)$, и найдите все значения n, при которых имеет место равенство.

3. [*M22*] Если допускаются равные ключи, то при сортировке трех элементов возможны 13 исходов:

$$\begin{array}{llll} K_1 = K_2 = K_3, & K_1 = K_2 < K_3, & K_1 = K_3 < K_2, & \\ K_2 = K_3 < K_1, & K_1 < K_2 = K_3, & K_2 < K_1 = K_3, & \\ K_3 < K_1 = K_2, & K_1 < K_2 < K_3, & K_1 < K_3 < K_2, & \\ K_2 < K_1 < K_3, & K_2 < K_3 < K_1, & K_3 < K_1 < K_2, & K_3 < K_2 < K_1. \end{array}$$

Обозначим через P_n число возможных исходов при сортировке n элементов, если допускаются равные ключи, так что $(P_0, P_1, P_2, P_3, P_4, P_5, \dots) = (1, 1, 3, 13, 75, 541, \dots)$. Докажите, что производящая функция $P(z) = \sum_{n\ge 0} P_n z^n/n!$ равна $1/(2 - e^z)$. *Указание.* Покажите, что

$$P_n = \sum_{k>0} \binom{n}{k} P_{n-k} \qquad \text{при } n > 0.$$

4. [*HM27*] (О. А. Гросс (O. A. Gross).) Найдите асимптотическое выражение для чисел P_n из упр. 3 при $n \to \infty$. [*Указание.* Рассмотрите разложение $\cot z$ на элементарные дроби.]

5. [*16*] Если допускаются равные ключи, то каждое сравнение может иметь не два, а три результата: $K_i < K_j$, $K_i = K_j$, $K_i > K_j$. В этом общем случае алгоритмы сортировки можно представлять в виде расширенных *тернарных* деревьев, в которых каждый внутренний узел $i:j$ имеет три поддерева (левое, среднее и правое), соответствующие трем возможным исходам сравнения.

Нарисуйте расширенное тернарное дерево, определяющее алгоритм сортировки для $n = 3$, если допускаются равные ключи. В вашем дереве должно быть 13 внешних узлов, соответствующих 13 возможным исходам, перечисленным в упр. 3.

▶ **6.** [*M22*] Пусть $S'(n)$ — минимальное число сравнений, необходимое для сортировки n элементов и выявления всех равенств между ключами, если каждое сравнение имеет три возможных результата, как в упр. 5. Нетрудно обобщить "теоретико-информационные" аргументы, приведенные в тексте раздела, и показать, что $S'(n) \ge \lceil \log_3 P_n \rceil$, где P_n — функция, проанализированная в упр. 3 и 4; докажите, что на самом деле $S'(n) = S(n)$.

7. [*20*] Нарисуйте расширенное тернарное дерево, как в упр. 5, для сортировки четырех элементов, если известно, что все ключи равны либо 0, либо 1. (Так, например, если $K_1 < K_2$ и $K_3 < K_4$, то понятно, что $K_1 = K_3$ и $K_2 = K_4$!) Добейтесь минимального числа сравнений в среднем, считая, что все 2^4 возможных исходных массивов равновероятны.

Обратите внимание на то, что должны быть проанализированы все имеющиеся варианты равенств; например, не прекращайте сортировку, если становится известно, что $K_1 \le K_2 \le K_3 \le K_4$.

8. [*26*] Нарисуйте расширенное тернарное дерево, как в упр. 7, для сортировки четырех элементов, если известно, что ключи — это либо -1, либо 0, либо $+1$. Добейтесь минимального числа сравнений в среднем, считая, что все 3^4 возможных исходных массивов равновероятны.

9. [*M20*] Каково минимальное число сравнений в наихудшем случае при сортировке n элементов, как в упр. 7, когда известно, что все элементы равны либо 0, либо 1?

▶ **10.** [*M25*] Каково минимальное *среднее* число сравнений при сортировке n элементов, как в упр. 7, когда известно, что все ключи равны либо 0, либо 1? Результат представьте в виде функции от n.

11. [*HM27*] Пусть $S_m(n)$ — минимальное число сравнений, необходимое в наихудшем случае для сортировки n элементов, как в упр. 5, причем известно, что все ключи принадлежат множеству $\{1, 2, \ldots, m\}$. [Таким образом, согласно упр. 6 $S_n(n) = S(n)$.] Докажите, что $S_m(n)$ приближается к $n \lg m + O(1)$ при фиксированном m и $n \to \infty$.

▶ **12.** [*M25*] (У. Г. Бурисиус (W. G. Bouricius), ок. 1954 г.) Предположим, что равные ключи могут встречаться, но наша цель — рассортировать элементы $\{K_1, K_2, \ldots, K_n\}$ таким образом, чтобы сформировать перестановку $a_1\, a_2 \ldots a_n$, удовлетворяющую условию $K_{a_1} \le K_{a_2} \le \cdots \le K_{a_n}$; нам не важно, имеет ли место равенство между элементами K_{a_i} и $K_{a_{i+1}}$.

Будем говорить, что дерево сравнений *сильно* сортирует последовательность ключей, если оно сортирует ее в вышеуказанном смысле, независимо от того, какой путь выбран в узлах $i{:}j$, для которых $K_i = K_j$. (Дерево бинарное, а не тернарное.)

a) Докажите, что дерево, не содержащее избыточных сравнений, сильно сортирует любую последовательность ключей тогда и только тогда, когда оно сортирует любую последовательность различных ключей.

b) Докажите, что дерево сравнений сильно сортирует любую последовательность ключей тогда и только тогда, когда оно сильно сортирует любую последовательность нулей и единиц.

13. [*M28*] Докажите утверждение (17).

14. [*M24*] Выразите сумму (19) в замкнутом виде.

15. [*M21*] Определите асимптотическое поведение функций $B(n)$ и $F(n)$ с точностью до $O(\log n)$. [*Указание.* Покажите, что в обоих случаях коэффициент при n содержит функцию, изображенную на рис. 37.)

16. [*HM26*] (Ф. Хванг (F. Hwang) и Шень Линь (S. Lin).) Докажите, что при $n \ge 22$ выполняется неравенство $F(n) > \lceil \lg n! \rceil$.

17. [*M20*] Докажите тождество (29).

18. [*20*] Предположим, что процедура, начало которой изображено на рис. 36, породила граф с эффективностью $12!/2^{29}$. Доказывает ли это, что $S(12) = 29$?

19. [*40*] Проведите эксперименты со следующим эвристическим правилом принятия решения, какую пару ключей сравнивать следующей при конструировании дерева сравнений. Пусть на каждой стадии сортировки ключей $\{K_1, \ldots, K_n\}$ число ключей, о которых на основании выполненных до сих пор сравнений известно, что они $\le K_i$, обозначается через u_i, а число ключей, о которых известно, что они $\ge K_i$, обозначается через v_i, $1 \le i \le n$. Перенумеруем ключи так, чтобы последовательность u_i/v_i стала возрастающей: $u_1/v_1 \le u_2/v_2 \le \cdots \le u_n/v_n$. Теперь сравним $K_i{:}K_{i+1}$, где i — индекс, минимизирующий

выражение $|u_i v_{i+1} - u_{i+1} v_i|$. (Хотя этот метод использует гораздо меньше информации, чем содержится в полной матрице сравнений, подобной (24), он, как оказывается, во многих случаях дает оптимальные результаты.)

▶ **20.** [*M26*] Докажите, что расширенное бинарное дерево имеет минимальную длину внешнего пути тогда и только тогда, когда существует такое число l, что все внешние узлы находятся на уровнях l и $l+1$ (или, быть может, только на уровне l).

21. [*M21*] *Высотой* расширенного бинарного дерева называется максимальный номер уровня, на котором есть внешние узлы. Пусть x — внутренний узел расширенного бинарного дерева; обозначим через $t(x)$ число внешних узлов-потомков узла x, а через $l(x)$ — корень левого поддерева узла x. Если x — внешний узел, то положим $t(x) = 1$. Докажите, что расширенное бинарное дерево имеет минимальную высоту среди всех бинарных деревьев с тем же числом узлов тогда и только тогда, когда для всех его внутренних узлов x выполняется неравенство

$$\bigl|t(x) - 2t(l(x))\bigr| \le 2^{\lceil \lg t(x) \rceil} - t(x).$$

22. [*M24*] Продолжение упр. 21. Докажите, что бинарное дерево имеет минимальную длину внешнего пути среди всех бинарных деревьев с тем же числом узлов тогда и только тогда, когда для всех его внутренних узлов x выполняются неравенства

$$\bigl|t(x) - 2t(l(x))\bigr| \le 2^{\lceil \lg t(x) \rceil} - t(x) \qquad \text{и} \qquad \bigl|t(x) - 2t(l(x))\bigr| \le t(x) - 2^{\lfloor \lg t(x) \rfloor}.$$

[Так, например, если $t(x) = 67$, то должно выполняться следующее: $t(l(x)) = 32$, 33, 34 или 35. Если нужно просто минимизировать высоту дерева, то согласно предыдущему упражнению достаточно, чтобы $3 \le t(l(x)) \le 64$.]

23. [*10*] В основном тексте раздела доказано, что среднее число сравнений, выполняемых любым методом сортировки n элементов, не может быть меньше $\lceil \lg n! \rceil \approx n \lg n$. Однако при сортировке методом вставок в несколько списков (программа 5.2.1M) затрачивается в среднем всего $O(n)$ машинных циклов. Чем это объясняется?

24. [*27*] (К. Пикар (C. Picard).) Постройте такое дерево сортировки для шести элементов, чтобы все его внешние узлы располагались на уровнях 10 и 11.

25. [*11*] Если бы существовала процедура сортировки семи элементов, при которой достигался бы минимум среднего числа сравнений, вычисляемый при помощи формулы (34), то сколько внешних узлов было бы на уровне 13 соответствующего дерева?

26. [*M42*] Найдите процедуру сортировки для семи элементов, минимизирующую среднее число выполняемых сравнений.

▶ **27.** [*20*] Пусть известно, что конфигурации $K_1 < K_2 < K_3$, $K_1 < K_3 < K_2$, $K_2 < K_1 < K_3$, $K_2 < K_3 < K_1$, $K_3 < K_1 < K_2$, $K_3 < K_2 < K_1$ встречаются с вероятностями соответственно .01, .25, .01, .24, .25, .24. Найдите дерево сравнений, которое бы сортировало такие три элемента с наименьшим средним числом сравнений.

28. [*40*] Напишите программу для MIX, которая сортирует 5 однословных ключей за минимально возможное время, после чего останавливается. (См. основные правила в начале раздела 5.2.)

29. [*M25*] (С. М. Чэйз (S. M. Chase).) Пусть $a_1 a_2 \ldots a_n$ — перестановка множества $\{1, 2, \ldots, n\}$. Докажите, что любой алгоритм, который распознает, является ли данная перестановка четной или нечетной (т. е. содержит ли она четное или нечетное число инверсий), и основанный исключительно на сравнениях элементов a, должен выполнить не менее $n \lg n$ сравнений, хотя он имеет всего два возможных исхода.

30. [*M23*] (*Оптимальная обменная сортировка.*) Любой алгоритм обменной сортировки в соответствии с определением, данным в разделе 5.2.2, можно представить в виде *дерева сравнений-обменов*, а именно — в виде структуры бинарного дерева, внутренние узлы которого имеют вид $i:j$, где $i < j$, и интерпретируются следующим образом: "Если $K_i \le K_j$, то продвинуться по левой ветви дерева; если $K_i > K_j$, то поменять местами записи i и j и продвинуться по правой ветви дерева". По достижении внешнего узла должны выполняться условия $K_1 \le K_2 \le \cdots \le K_n$. Таким образом, дерево сравнений-обменов отличается от дерева сравнений тем, что оно описывает не только операции сравнения, но и операции перемещения данных.

Обозначим через $S_e(n)$ минимальное число сравнений-обменов, необходимых в наихудшем случае для сортировки элементов при помощи дерева сравнений-обменов. Докажите, что $S_e(n) \le S(n) + n - 1$.

31. [*M38*] Продолжение упр. 30. Докажите, что $S_e(5) = 8$.

32. [*M42*] Продолжение упр. 31. Исследуйте значения функции $S_e(n)$ при малых $n > 5$.

33. [*M30*] (Т. Н. Хиббард (T. N. Hibbard).) *Вещественнозначным деревом поиска* порядка x с разрешением δ называется расширенное бинарное дерево, каждый узел которого содержит неотрицательное действительное значение, такое, что: (i) значение в любом внешнем узле $\le \delta$; (ii) значение в любом внутреннем узле не превышает суммы значений двух его потомков; (iii) значение в корне равно x. *Длина взвешенного пути* такого дерева определяется как сумма по всем внешним узлам номеров уровней этих узлов, умноженных на содержащиеся в них значения.

Докажите, что вещественнозначное дерево поиска порядка x с разрешением 1 имеет длину взвешенного пути, минимальную среди всех таких деревьев того же порядка и с тем же разрешением тогда и только тогда, когда в (ii) имеет место равенство и для всех пар значений x_0 и x_1, принадлежащих узлам-братьям, выполняются следующие условия: (iv) не существует целого числа $k \ge 0$, такого, что $x_0 < 2^k < x_1$ или $x_1 < 2^k < x_0$; (v) $\lceil x_0 \rceil - x_0 + \lceil x_1 \rceil - x_1 < 1$. (В частности, если x — целое число, то из условия (v) следует, что все значения в дереве — целые, а условие (iv) эквивалентно результату упр. 22.)

Докажите также, что соответствующая минимальная длина взвешенного пути равна $x\lceil \lg x \rceil + \lceil x \rceil - 2^{\lceil \lg x \rceil}$.

34. [*M50*] Определите точные значения функции $S(n)$ для бесконечного множества аргументов n.

35. [*49*] Определите точное значение функции $S(13)$.

36. [*M50*] (С. С. Кислицын, 1968.) Докажите или опровергните следующее утверждение: любой ориентированный ациклический граф G, в котором $T(G) > 1$, имеет две вершины, u и v, такие, что диграфы G_1 и G_2, полученные из G в результате добавления дуг $u \leftarrow v$ и $u \rightarrow v$, также ацикличны и удовлетворяют условию $1 \le T(G_1)/T(G_2) \le 2$. (Таким образом, $T(G_1)/T(G)$ всегда лежит между $\frac{1}{3}$ и $\frac{2}{3}$ при некоторых u и v.)

*5.3.2. Слияние с минимальным числом сравнений

Рассмотрим теперь вопрос, имеющий отношение к предыдущему разделу: каков наилучший способ слияния упорядоченного множества m элементов с упорядоченным множеством n элементов? Обозначим элементы сливаемых множеств

$$A_1 < A_2 < \cdots < A_m \qquad \text{и} \qquad B_1 < B_2 < \cdots < B_n. \tag{1}$$

Как и в разделе 5.3.1, будем предполагать, что все $m + n$ элементов различны. Элементы A_k среди элементов B_k могут располагаться $\binom{m+n}{m}$ способами. Таким

образом, из анализа, которым мы воспользовались в задаче о сортировке, следует, что необходимо выполнить, по крайней мере,

$$\left\lceil \lg\binom{m+n}{m} \right\rceil \tag{2}$$

сравнений. Если положить $m = \alpha n$ и устремить $n \to \infty$, оставив α неизменным, то по формуле Стирлинга

$$\lg\binom{\alpha n + n}{\alpha n} = n\big((1+\alpha)\lg(1+\alpha) - \alpha\lg\alpha\big) - \tfrac{1}{2}\lg n + O(1). \tag{3}$$

Обычная процедура слияния (алгоритм 5.2.4M) выполняет в худшем случае $m+n-1$ сравнений.

Обозначим через $M(m,n)$ функцию, аналогичную $S(n)$, а именно минимальное число сравнений, заведомо достаточное для слияния m элементов с n элементами. Из только что сделанного вывода следует, что

$$\left\lceil \lg\binom{m+n}{m} \right\rceil \le M(m,n) \le m+n-1 \qquad \text{при } m,n \ge 1. \tag{4}$$

Формула (3) показывает, насколько далеко могут отстоять друг от друга нижняя и верхняя оценки. При $\alpha = 1$ (т. е. $m = n$) нижняя оценка равна $2n - \frac{1}{2}\lg n + O(1)$, так что обе оценки — величины одного порядка, но разность между ними может быть сколь угодно велика. При $\alpha = 0.5$ (т. е. $m = \frac{1}{2}n$) нижняя оценка равна

$$\tfrac{3}{2}n(\lg 3 - \tfrac{2}{3}) + O(\log n),$$

что составляет примерно $\lg 3 - \frac{2}{3} \approx 0.918$ от верхней оценки. С убыванием α разница между верхней и нижней оценками все увеличивается, поскольку стандартный алгоритм слияния разработан, главным образом, для массивов с $m \approx n$.

При $m = n$ задача о слиянии имеет весьма простое решение; слишком грубой оказывается не верхняя, а *нижняя* оценка в (4). Следующую теорему независимо доказали Р. Л. Грэхем (R. L. Graham) и Р. М. Карп (R. M. Karp) примерно в 1968 году.

Теорема М. $M(m,m) = 2m-1$ *при* $m \ge 1$.

Доказательство. Рассмотрим какой-нибудь алгоритм, который осуществляет слияние элементов $A_1 < \cdots < A_m$ с $B_1 < \cdots < B_m$. При сравнении элементов $A_i : B_j$ выберем ветвь $A_i < B_j$, если $i < j$, и ветвь $A_i > B_j$, если $i \ge j$. Слияние должно завершиться конфигурацией

$$B_1 < A_1 < B_2 < A_2 < \cdots < B_m < A_m, \tag{5}$$

поскольку она согласуется со всеми выбранными ветвями. И каждое из $2m-1$ сравнений $B_1 : A_1$, $A_1 : B_2$, $B_2 : A_2$, $\ldots$, $B_m : A_m$ должно быть выполнено явно, иначе найдется по меньшей мере две конфигурации, не противоречащие известным фактам. Если бы мы, например, не сравнили A_1 с B_2, то конфигурация

$$B_1 < B_2 < A_1 < A_2 < \cdots < B_m < A_m$$

была бы неотличима от (5). ∎

Несложная модификация этого доказательства дает аналогичную формулу

$$M(m, m+1) = 2m \qquad \text{при } m \geq 0. \tag{6}$$

Определение нижних оценок. Теорема М показывает, что "теоретико-информационная" нижняя оценка (2) может сколь угодно далеко отстоять от истинной нижней границы; таким образом, метод доказательства теоремы М дает нам еще один способ нахождения нижних оценок. Такой метод доказательства часто рассматривается как порождение *соперника*, советы которого принуждают алгоритм работать как можно медленнее. Когда алгоритм слияния решает сравнить элементы $A_i : B_j$, соперник так определяет судьбу сравнения, что вынуждает алгоритм избрать наиболее трудный путь. Если бы мы смогли придумать подходящего соперника, то смогли бы убедиться в том, что всякий правильный алгоритм слияния должен выполнить довольно мало сравнений.

Мы будем использовать *соперников с ограниченными возможностями*, воздействие которых лимитировано заранее заданными результатами некоторых сравнений. В методе слияния, находящемся под воздействием соперника с ограниченными возможностями, ограничения считаются неизвестными, и поэтому выполняются все необходимые сравнения даже в том случае, когда их результат предопределен. Например, в доказательстве теоремы М мы ограничили все результаты сравнений условием (5), тем не менее в алгоритме слияния нельзя воспользоваться этим обстоятельством, чтобы избежать хотя бы одного сравнения.

Ограничения, которые мы будем использовать в следующем ниже анализе, относятся к левому и правому концам массивов. Левые ограничения обозначаются следующими символами:

. (нет ограничения слева),

\ (результаты всех сравнений не должны противоречить условию $A_1 < B_1$),

/ (результаты всех сравнений не должны противоречить условию $A_1 > B_1$).

Аналогично правые ограничения обозначаются следующими символами:

. (нет ограничения справа),

\ (результаты всех сравнений не должны противоречить условию $A_m < B_n$),

/ (результаты всех сравнений не должны противоречить условию $A_m > B_n$).

Существует девять типов соперников, обозначаемых символами $\lambda M \rho$, где λ — левое ограничение, а ρ — правое. Например, соперник $\backslash M \backslash$ должен говорить, что $A_1 < B_j$ и $A_i < B_n$; соперник $.M.$ не подчиняется никаким ограничениям. При некоторых малых значениях m и n может не существовать соперников с ограниченными возможностями некоторых типов; при $m = 1$, очевидно, не может быть соперника $\backslash M /$.

Займемся теперь построением весьма сложного, но чрезвычайно коварного соперника для слияний. Он не всегда порождает оптимальные результаты, но дает нижние оценки, которые охватывают множество интересных случаев. Предположим, заданы m и n, а также левые и правые ограничения λ и ρ. Пусть соперника спрашивают, какой из двух элементов (A_i или B_j) больше. Соперник может, вообще говоря, применить 6 стратегий приведения задачи к случаю меньшего значения $m + n$.

Стратегия $\mathrm{A}(k,l)$ *для* $i \le k \le m$ *и* $1 \le l \le j$. Ответить, что $A_i < B_j$, и потребовать, чтобы последующие операции осуществляли слияния $\{A_1,\ldots,A_k\}$ с $\{B_1,\ldots,B_{l-1}\}$ и $\{A_{k+1},\ldots,A_m\}$ с $\{B_l,\ldots,B_n\}$. Тогда последующие сравнения $A_p{:}B_q$ дадут такие результаты: $A_p < B_q$, если $p \le k$ и $q \ge l$; $A_p > B_q$, если $p > k$ и $q < l$; они будут управляться соперником $(k, l-1, \lambda, .)$, если $p \le k$ и $q < l$, и соперником $(m-k, n+1-l, ., \rho)$, если $p > k$ и $q \ge l$.

Стратегия $\mathrm{B}(k,l)$ *для* $i \le k \le m$ *и* $1 \le l < j$. Ответить, что $A_i < B_j$, и потребовать, чтобы последующие операции осуществляли слияния $\{A_1,\ldots,A_k\}$ с $\{B_1,\ldots,B_l\}$ и $\{A_{k+1},\ldots,A_m\}$ с $\{B_l,\ldots,B_n\}$ при условии $A_k < B_l < A_{k+1}$. (Обратите внимание на то, что B_l присутствует в обоих списках, подлежащих слиянию. Условие $A_k < B_l < A_{k+1}$ обеспечивает такое положение, при котором слияние одной пары массивов не может дать никакой информации, которая бы помогла при слиянии другой пары.) Тогда последующие сравнения $A_p{:}B_q$ дадут такие результаты: $A_p < B_q$, если $p \le k$ и $q \ge l$; $A_p > B_q$, если $p > k$ и $q \le l$; они будут управляться соперником $(k, l, \lambda, \backslash)$, если $p \le k$ и $q \le l$, и соперником $(m-k, n+1-l, /, \rho)$, если $p > k$ и $q \ge l$.

Стратегия $\mathrm{C}(k,l)$ *для* $i < k \le m$ *и* $1 \le l \le j$. Ответить, что $A_i < B_j$, и потребовать, чтобы последующие операции осуществляли слияния $\{A_1,\ldots,A_k\}$ с $\{B_1,\ldots,B_{l-1}\}$ и $\{A_k,\ldots,A_m\}$ с $\{B_l,\ldots,B_n\}$ при условии, что $B_{l-1} < A_k < B_l$. (Эта стратегия аналогична стратегии B, но массивы A и B меняются ролями.)

Стратегия $\mathrm{A}'(k,l)$ *для* $1 \le k \le i$ *и* $j \le l \le n$. Ответить, что $A_i > B_j$, и потребовать, чтобы последующие операции осуществляли слияния $\{A_1,\ldots,A_{k-1}\}$ с $\{B_1,\ldots,B_l\}$ и $\{A_k,\ldots,A_m\}$ с $\{B_{l+1},\ldots,B_n\}$. (Эта стратегия аналогична стратегии A.)

Стратегия $\mathrm{B}'(k,l)$ *для* $1 \le k \le i$ *и* $j < l \le n$. Ответить, что $A_i > B_j$, и потребовать, чтобы последующие операции осуществляли слияния $\{A_1,\ldots,A_{k-1}\}$ с $\{B_1,\ldots,B_l\}$ и $\{A_k,\ldots,A_m\}$ с $\{B_l,\ldots,B_n\}$ при условии $A_{k-1} < B_l < A_k$. (Эта стратегия аналогична стратегии B.)

Стратегия $\mathrm{C}'(k,l)$ *для* $1 \le k < i$ *и* $j \le l \le n$. Ответить, что $A_i > B_j$, и потребовать, чтобы последующие операции осуществляли слияния $\{A_1,\ldots,A_k\}$ с $\{B_1,\ldots,B_l\}$ и $\{A_k,\ldots,A_m\}$ с $\{B_{l+1},\ldots,B_n\}$ при условии $B_l < A_k < B_{l+1}$. (Эта стратегия аналогична стратегии C.)

В случаях, которые перечислены ниже, приведенные выше стратегии не могут применяться из-за налагаемых ограничений.

Стратегия	Не должна применяться, если
$\mathrm{A}(k,1)$, $\mathrm{B}(k,1)$, $\mathrm{C}(k,1)$	$\lambda = /$
$\mathrm{A}'(1,l)$, $\mathrm{B}'(1,l)$, $\mathrm{C}'(1,l)$	$\lambda = \backslash$
$\mathrm{A}(m,l)$, $\mathrm{B}(m,l)$, $\mathrm{C}(m,l)$	$\rho = /$
$\mathrm{A}'(k,n)$, $\mathrm{B}'(k,n)$, $\mathrm{C}'(k,n)$	$\rho = \backslash$

Обозначим через $\lambda M\rho(m,n)$ максимальную нижнюю оценку, которую можно получить при помощи соперника из описанного выше класса. Если первое сравнение есть $A_i{:}B_j$, то каждая стратегия, если она применима, дает неравенства, связывающие эти девять функций, а именно:

$$
\begin{aligned}
&\mathrm{A}(k,l): && \lambda M\rho(m,n) \geq 1 + \lambda M.(k,l-1) + .M\rho(m-k,n+1-l);\\
&\mathrm{B}(k,l): && \lambda M\rho(m,n) \geq 1 + \lambda M\backslash(k,l) + /M\rho(m-k,n+1-l);\\
&\mathrm{C}(k,l): && \lambda M\rho(m,n) \geq 1 + \lambda M/(k,l-1) + \backslash M\rho(m+1-k,n+1-l);\\
&\mathrm{A}'(k,l): && \lambda M\rho(m,n) \geq 1 + \lambda M.(k-1,l) + .M\rho(m+1-k,n-l);\\
&\mathrm{B}'(k,l): && \lambda M\rho(m,n) \geq 1 + \lambda M\backslash(k-1,l) + /M\rho(m+1-k,n+1-l);\\
&\mathrm{C}'(k,l): && \lambda M\rho(m,n) \geq 1 + \lambda M/(k,l) + \backslash M\rho(m+1-k,n-l).
\end{aligned}
$$

Для фиксированных i и j соперник примет ту стратегию, которая максимизирует нижнюю оценку, задаваемую правыми частями неравенств, если k и l лежат в пределах, определенных для i и j. Таким образом, $\lambda M\rho(m,n)$ есть минимум этих нижних оценок по всем $1 \leq i \leq m$ и $1 \leq j \leq n$. Если m или n равно 0, то и значение функции $\lambda M\rho(m,n)$ равно 0.

Пусть, например, $m = 2$ и $n = 3$, а наш соперник не ограничен. Если первым выполняется сравнение $A_1 : B_1$, то соперник может принять стратегию $\mathrm{A}'(1,1)$, в результате чего потребуется еще $.M.(0,1) + .M.(2,2) = 3$ сравнения. Если первым выполняется сравнение $A_1 : B_3$, то он может выбрать стратегию $\mathrm{B}(1,2)$ и тогда потребуется еще $.M\backslash(1,2) + /M.(1,2) = 4$ сравнения. Независимо от того, какое сравнение $A_i : B_j$ было сделано первым, соперник гарантирует выполнение еще, по крайней мере, трех сравнений. Следовательно, $.M.(2,3) = 4$.

Не так просто выполнить эти вычисления вручную, но компьютер позволяет довольно быстро получить таблицу значений функций $\lambda M\rho$. Эти функции обладают некоторыми очевидными свойствами симметрии

$$/M.(m,n) = .M\backslash(m,n) = \backslash M.(n,m) = .M/(n,m), \tag{7}$$

позволяющими свести наши девять функций всего лишь к четырем:

$$.M.(m,n), \qquad /M.(m,n), \qquad /M\backslash(m,n) \qquad \text{и} \qquad /M/(m,n).$$

В табл. 1 приведены значения для всех $m, n \leq 10$. Наш соперник для слияний определен таким образом, что

$$.M.(m,n) \leq M(m,n) \qquad \text{при всех} \qquad m,n \geq 0. \tag{8}$$

Данное соотношение содержит в качестве частного случая теорему M, поскольку при $|m-n| \leq 1$ наш соперник изберет простую стратегию этой теоремы.

Рассмотрим теперь несколько простых соотношений, которым удовлетворяет функция M:

$$M(m,n) = M(n,m); \tag{9}$$

$$M(m,n) \leq M(m,n+1); \tag{10}$$

$$M(k+m,n) \leq M(k,n) + M(m,n); \tag{11}$$

$$M(m,n) \leq \max\bigl(M(m,n-1)+1,\ M(m-1,n)+1\bigr) \quad \text{при } m \geq 1,\ n \geq 1; \tag{12}$$

$$M(m,n) \leq \max\bigl(M(m,n-2)+1,\ M(m-1,n)+2\bigr) \quad \text{при } m \geq 1,\ n \geq 2. \tag{13}$$

Соотношение (12) следует из обычной процедуры слияния, если начать со сравнения элементов $A_1 : B_1$. Соотношение (13) выводится аналогично, только сначала сравниваются $A_1 : B_2$; если $A_1 > B_2$, то нужно еще $M(m,n-2)$ сравнений, если же $A_1 < B_2$, то можно вставить A_1 в соответствующее место и слить $\{A_2, \ldots, A_m\}$ с $\{B_1, \ldots, B_n\}$.

Таблица 1

НИЖНИЕ ОЦЕНКИ ДЛЯ СЛИЯНИЙ, ВЫПОЛНЕННЫХ ПРИ УЧАСТИИ СОПЕРНИКА

$.M.(m,n)$

$m \backslash n$	1	2	3	4	5	6	7	8	9	10
1	1	2	2	3	3	3	3	4	4	4
2	2	3	4	5	5	6	6	6	7	7
3	2	4	5	6	7	7	8	8	9	9
4	3	5	6	7	8	9	10	10	11	11
5	3	5	7	8	9	10	11	12	12	13
6	3	6	7	9	10	11	12	13	14	15
7	3	6	8	10	11	12	13	14	15	16
8	4	6	8	10	12	13	14	15	16	17
9	4	7	9	11	12	14	15	16	17	18
10	4	7	9	11	13	15	16	17	18	19

$/M.(m,n)$

$m \backslash n$	1	2	3	4	5	6	7	8	9	10
1	1	2	2	3	3	3	3	4	4	4
2	1	3	4	4	5	5	6	6	7	7
3	1	3	5	6	7	7	8	8	9	9
4	1	4	5	7	8	9	9	10	10	11
5	1	4	6	8	9	10	11	12	12	13
6	1	4	6	8	10	11	12	13	14	14
7	1	4	7	9	10	12	13	14	15	16
8	1	5	7	9	11	13	14	15	16	17
9	1	5	8	10	11	13	15	16	17	18
10	1	5	8	10	12	14	15	17	18	19

$/M\backslash(m,n)$

$m \backslash n$	1	2	3	4	5	6	7	8	9	10
1	$-\infty$	2	2	3	3	3	4	4	4	4
2	$-\infty$	2	4	4	5	5	6	6	7	7
3	$-\infty$	2	4	6	6	7	8	8	8	9
4	$-\infty$	2	5	6	8	8	9	10	10	11
5	$-\infty$	2	5	7	8	10	10	11	12	13
6	$-\infty$	2	5	7	9	10	12	13	14	14
7	$-\infty$	2	5	8	10	11	12	14	15	16
8	$-\infty$	2	6	8	10	12	13	15	16	17
9	$-\infty$	2	6	9	10	12	14	16	17	18
10	$-\infty$	2	6	9	11	13	15	16	18	19

$/M/(m,n)$

$m \backslash n$	1	2	3	4	5	6	7	8	9	10
1	1	1	1	1	1	1	1	1	1	1
2	1	3	3	4	4	4	4	5	5	5
3	1	3	5	5	6	6	7	7	8	8
4	1	4	5	7	7	8	9	9	9	10
5	1	4	6	7	9	9	10	11	11	12
6	1	4	6	8	9	11	11	12	13	14
7	1	4	7	9	10	11	13	14	15	15
8	1	5	7	9	11	12	14	15	16	17
9	1	5	8	9	11	13	15	16	17	18
10	1	5	8	10	12	14	15	17	18	19

Если перейти к более общему случаю, выполнив для этого сначала сравнение $A_1 : B_k$, а затем использовав двоичный поиск в случае $A_1 < B_k$, увидим, что при $m \geq 1$ и $n \geq k$ выполняется неравенство

$$M(m,n) \leq \max\bigl(M(m,n-k)+1,\ M(m-1,n)+1+\lceil \lg k \rceil\bigr). \tag{14}$$

Оказывается, $M(m,n) = .M.(m,n)$ при всех $m,n \leq 10$, так что в табл. 1 в действительности приведены оптимальные значения для слияний. Это можно доказать, применяя соотношения (9)–(14), а также специальные построения для $(m,n) = (2,8)$, $(3,6)$ и $(5,9)$, которые приводятся в упр. 8–10.

С другой стороны, такой соперник не всегда дает наилучшую возможную нижнюю оценку. Простейший пример: $m = 3$, $n = 11$, когда $.M.(3,11) = 9$, а $M(3,11) = 10$. Чтобы понять, где же в этом случае наш соперник "промахнулся", нужно изучить мотивы, на которых основаны его решения; при дальнейшем тщательном исследовании обнаруживается, что если $(i,j) \neq (2,6)$, то соперник может отыскать стратегии, требующие 10 сравнений; если же $(i,j) = (2,6)$, то ни одна стратегия не превосходит стратегию A$(2,4)$, которая приводит к нижней оценке $1 + .M.(2,3) + .M.(1,8) = 9$. Необходимо, но не достаточно, чтобы процесс заканчивался слиянием $\{A_1, A_2\}$ с $\{B_1, B_2, B_3\}$ и $\{A_3\}$ с $\{B_4, \ldots, B_{11}\}$; таким образом, нижняя оценка в этом случае оказывается неточной.

Аналогично можно показать, что $.M.(2,38) = 10$, в то время как $M(2,38) = 11$, и т. д. Значит, наш соперник не достаточно искусен, чтобы справиться со случаем $m = 2$. Однако существует бесконечный класс значений, для которых он работает безупречно.

Теорема К. $M(m, m{+}2) = 2m + 1$ *при* $m \geq 2$;
$M(m, m{+}3) = 2m + 2$ *при* $m \geq 4$;
$M(m, m{+}4) = 2m + 3$ *при* $m \geq 6$.

Доказательство. На самом деле мы можем доказывать эти соотношения, заменив M на $.M.$; при малых m эти результаты были получены на компьютере, поэтому можно предполагать, что m достаточно велико. Можно также считать, что первое сравнение есть $A_i{:}B_j$, где $i \leq \lceil m/2 \rceil$. Если $j \leq i$, то воспользуемся стратегией $\mathrm{A}'(i, i)$; тогда получим

$$.M.(m, m{+}d) \geq 1 + .M.(i{-}1, i) + .M.(m{+}1{-}i, m{+}d{-}i) = 2m + d - 1,$$

применив индукцию по d, $d \leq 4$. Если $j > i$, то воспользуемся стратегией $\mathrm{A}(i, i{+}1)$; применив индукцию по m, получим

$$.M.(m, m{+}d) \geq 1 + .M.(i, i) + .M.(m{-}i, m{+}d{-}i) = 2m + d - 1. \quad \blacksquare$$

Первые два утверждения теоремы К получили Ф. Хванг и Ш. Линь в 1969 году. Спустя несколько лет Пол Стокмайер (Paul Stockmeyer) и Фрэнсис Яо (F. F. C. Yao) показали, что некоторые свойства этих формул можно распространить и на более общий случай. В частности, нижние оценки, выведенные стратегиями соперника, достаточно удовлетворительны для того, чтобы обеспечить значения $M(m, m{+}d) = 2m + d - 1$ при $m \geq 2d - 2$. [*SICOMP* **9** (1980), 85–90.]

Верхние оценки. Рассмотрим теперь *верхние* оценки функции $M(m, n)$. Хорошие верхние оценки соответствуют эффективным алгоритмам слияния.

При $m = 1$ задача слияния эквивалентна задаче вставки, когда имеется $n + 1$ мест между элементами $B_1, \ldots, B_n$, куда может попасть элемент A_1. В этом случае нетрудно видеть, что *любое* расширенное бинарное дерево с $n + 1$ внешними узлами есть дерево для некоторого метода слияния (см. упр. 2)! Следовательно, можно выбрать оптимальное бинарное дерево, реализовав теоретико-информационную нижнюю оценку

$$1 + \lfloor \lg n \rfloor = M(1, n) = \lceil \lg(n + 1) \rceil. \tag{15}$$

Разумеется, бинарный поиск (раздел 6.2.1) — простейший способ, позволяющий достичь этого значения.

Случай $m = 2$ чрезвычайно интересен, но анализировать его гораздо сложнее. Этот анализ полностью выполнен Р. Л. Грэхемом, Ф. К. Хуаном и Ш. Линем, которые вывели формулу для общего случая (см. упр. 11–13):

$$M(2, n) = \lceil \lg \tfrac{7}{12}(n + 1) \rceil + \lceil \lg \tfrac{14}{17}(n + 1) \rceil. \tag{16}$$

Мы видели, что при $m = n$ оптимальна обычная процедура слияния, а при $m = 1$ оптимальна довольно сильно отличающаяся от нее процедура бинарного поиска. Нам же нужен некоторый промежуточный метод, объединяющий в себе лучшие черты алгоритмов обычного слияния и бинарного поиска. Формула (14) наводит на мысль о следующем алгоритме (F. K. Hwang и S. Lin [*SICOMP* **1** (1972), 31–39]).

Алгоритм Н (*Бинарное слияние*).

Н1. Если m или n равно 0, прекратить выполнение алгоритма. В противном случае, если $m > n$, установить $t \leftarrow \lfloor \lg(m/n) \rfloor$ и перейти к шагу Н4. В противном случае установить $t \leftarrow \lfloor \lg(n/m) \rfloor$.

Н2. Сравнить $A_m : B_{n+1-2^t}$. Если A_m меньше, то установить $n \leftarrow n - 2^t$ и вернуться к шагу Н1.

Н3. Используя бинарный поиск (который требует ровно t дополнительных сравнений), вставить A_m в нужное место среди $\{B_{n+1-2^t}, \ldots, B_n\}$. Если k максимальное и такое, что $B_k < A_m$, установить $m \leftarrow m-1$ и $n \leftarrow k$. Вернуться к шагу Н1.

Н4. (Шаги Н4 и Н5 подобны Н2 и Н3, но переменные m и n, A и B меняются ролями.) Если $B_n < A_{m+1-2^t}$, установить $m \leftarrow m - 2^t$ и вернуться к шагу Н1.

Н5. Вставить B_n в нужное место среди элементов A. Если k максимальное и такое, что $A_k < B_n$, установить $m \leftarrow k$ и $n \leftarrow n - 1$. Вернуться к шагу Н1. ▮

В качестве примера работы этого алгоритма в табл. 2 показан процесс слияния трех ключей {087, 503, 512} с тринадцатью ключами {061, 154, ..., 908}; в этом примере выполняется восемь сравнений. Элементы, сравниваемые на каждом шаге алгоритма, выделены полужирным шрифтом.

Таблица 2

ПРИМЕР ПРИМЕНЕНИЯ МЕТОДА БИНАРНОГО СЛИЯНИЯ

A			*B*														Вывод		
087	503	**512**	061	154	170	275	426	509	612	653	677	**703**	765	897	908				
087	503	**512**	061	154	170	275	426	509	612	**653**	677				703	765	897	908	
087	503	**512**	061	154	170	275	426	**509**	612				653	677	703	765	897	908	
087	503	**512**	061	154	170	275	426	509	**612**				653	677	703	765	897	908	
087	**503**		061	154	170	275	**426**	509			512	612	653	677	703	765	897	908	
087	**503**		061	154	170	275	426	**509**			512	612	653	677	703	765	897	908	
087			061	**154**	170	275	426		503	509	512	612	653	677	703	765	897	908	
087			**061**		154	170	275	426	503	509	512	612	653	677	703	765	897	908	
			061	087	154	170	275	426	503	509	512	612	653	677	703	765	897	908	

Пусть $H(m,n)$ — максимальное число сравнений, выполняемых алгоритмом Хуана и Линя. Чтобы вычислить $H(m,n)$, можно предположить, что $k = n$ на шаге Н3 и $k = m$ на шаге Н5, поскольку при помощи индукции по m нетрудно доказать, что $H(m-1,n) \le H(m-1,n+1)$ при всех $n \ge m-1$. Таким образом, при $m \le n$ имеем

$$H(m,n) = \max\bigl(M(m, n-2^t)+1,\ H(m-1,n)+t+1\bigr) \tag{17}$$

при $2^t m \le n < 2^{t+1}m$. Заменим n на $2n+\epsilon$ ($\epsilon = 0$ или 1) и получим

$$H(m, 2n+\epsilon) = \max\bigl(H(m, 2n+\epsilon-2^{t+1}) + 1,\ H(m-1, 2n+\epsilon)+t+2\bigr)$$

при $2^t m \le n < 2^{t+1}m$. Отсюда вытекает, если применить индукцию по n, что

$$H(m, 2n+\epsilon) = H(m,n) + m \qquad \text{при } m \le n \text{ и } \epsilon = 0 \text{ или } 1. \tag{18}$$

Легко видеть также, что $H(m,n) = m + n - 1$, если $m \le n < 2m$. Следовательно, многократное применение формулы (18) даст нам общую формулу

$$H(m,n) = m + \lfloor n/2^t \rfloor - 1 + tm \quad \text{при} \quad m \le n, \quad t = \lfloor \lg(n/m) \rfloor. \tag{19}$$

Отсюда следует, что $H(m,n) \le H(m,n{+}1)$ для всех $n \ge m$, что подтверждает нашу гипотезу, полученную по индукции применительно к шагу H3.

Полагая $m = \alpha n$ и $\theta = \lg(n/m) - t$, найдем

$$H(\alpha n, n) = \alpha n(1 + 2^\theta - \theta - \lg\alpha) + O(1) \tag{20}$$

при $n \to \infty$. Из формул 5.3.1–(36) известно, что $1.9139 < 1 + 2^\theta - \theta \le 2$. Следовательно, (20) можно сравнить с теоретико-информационной нижней оценкой (3). Хуан и Линь доказали (см. упр. 17), что

$$H(m,n) < \left\lceil \lg\binom{m+n}{m} \right\rceil + \min(m,n). \tag{21}$$

Алгоритм бинарного слияния Хуана-Линя не всегда оптимален, но обладает тем неоценимым достоинством, что его довольно легко запрограммировать. Он сводится к "децентрированному бинарному поиску" при $m = 1$ и к обычной процедуре слияния при $m \approx n$, так что это золотая середина между двумя указанными методами. Кроме того, во многих случаях он *является* оптимальным (см. упр. 16). Дальнейшее совершенствование алгоритма описано в работах F. K. Hwang, D. N. Deutsch, *JACM* **20** (1973), 148–159; G. K. Manacher, *JACM* **26** (1979), 434–440; C. Christen, *FOCS* **19** (1978), 259–266. В последней из них Кристен (Christen) описал процедуру слияния, названную им *forward-testing-backward-insertion* (просмотр впереди, вставка позади), которая сокращает число сравнений примерно на $m/3$ по сравнению с алгоритмом H при $n/m \to \infty$. Более того, метод Кристена обеспечивает нижнюю оценку $.M.(m,n) = \lfloor (11m + n - 3)/4 \rfloor$, если $5m - 3 \le n \le 7m + 2[m \text{ even}]$; следовательно, при этих условиях он оптимален.

Формула (18) наводит на мысль о том, что и сама функция M, быть может, удовлетворяет неравенству

$$M(m,n) \le M(m, \lfloor n/2 \rfloor) + m. \tag{22}$$

Это и в самом деле так (см. упр. 19). Таблицы значений функции $M(m,n)$ позволяют предположить, что, возможно, имеют место и другие соотношения, такие, как

$$M(m{+}1,n) \ge 1 + M(m,n) \ge M(m,n{+}1) \qquad \text{при } m \le n; \tag{23}$$

$$M(m{+}1, n+1) \ge 2 + M(m,n); \tag{24}$$

но в настоящее время не известно никаких доказательств этих неравенств.

УПРАЖНЕНИЯ

1. [*15*] Найдите любопытное соотношение, которое связывает функцию $M(m,n)$ и функцию S, определенную в разделе 5.3.1. [*Указание.* Рассмотрите $S(m+n)$.]

▸ **2.** [*22*] При $m = 1$ любой алгоритм слияния, не содержащий избыточных сравнений, определяет расширенное бинарное дерево с $\binom{m+n}{m} = n{+}1$ внешними узлами. Докажите, что верно и обратное, т. е. каждому расширенному бинарному дереву соответствует некоторый алгоритм слияния с $m = 1$.

3. [*M24*] Докажите, что $.M.(1,n) = M(1,n)$ при всех n.

4. [*M42*] Справедливо ли неравенство $.M.(m,n) \geq \lceil \lg \binom{m+n}{m} \rceil$ при всех m и n?

5. [*M30*] Докажите, что $.M.(m,n) \leq .M\backslash(m,n+1)$.

6. [*M26*] Сформулированное выше доказательство теоремы К требует проверки на компьютере большого числа случаев. Каким образом можно резко сократить число таких случаев?

7. [*21*] Докажите неравенство (11).

▶ **8.** [*24*] Докажите, что $M(2,8) \leq 6$. Для этого придумайте такой алгоритм слияния двух элементов с восемью другими, который бы выполнял не более шести сравнений.

9. [*27*] Докажите, что три элемента можно слить с шестью элементами не более чем за семь шагов.

10. [*33*] Докажите, что пять элементов можно слить с девятью не более чем за двенадцать шагов. [*Указание.* Опыт введения соперника подсказывает, что начать нужно со сравнения $A_1 : B_2$, а затем, если $A_1 < B_2$, попытаться сравнить $A_5 : B_8$.]

11. [*M40*] (Ф. Хуан и Ш. Линь.) Пусть $g_{2k} = \lfloor 2^k \frac{17}{14} \rfloor$, $g_{2k+1} = \lfloor 2^k \frac{12}{7} \rfloor$ при $k \geq 0$, так что $(g_0, g_1, g_2, \ldots) = (1,1,2,3,4,6,9,13,19,27,38,54,77,\ldots)$. Докажите, что для слияния двух элементов с g_t элементами требуется в худшем случае более чем t сравнений, однако слить два элемента с $g_t - 1$ можно не более чем за t шагов. [*Указание.* Покажите, что если $n = g_t$ или $n = g_t - 1$ и если нам нужно слить $\{A_1, A_2\}$ с $\{B_1, B_2, \ldots, B_n\}$ за t сравнений, то наилучшее первое сравнение — это $A_2 : B_{g_{t-1}}$.]

12. [*M21*] Пусть $R_n(i,j)$ — наименьшее число сравнений, необходимое для сортировки различных объектов $\{\alpha, \beta, X_1, \ldots, X_n\}$, если заданы соотношения $\alpha < \beta$, $X_1 < X_2 < \cdots < X_n$, $\alpha < X_{i+1}$, $\beta > X_{n-j}$. (Условие $\alpha < X_{i+1}$ или $\beta > X_{n-j}$ теряет смысл, если $i \geq n$ или $j \geq n$. Поэтому $R_n(n,n) = M(2,n)$.)

Ясно, что $R_n(0,0) = 0$. Докажите, что

$$R_n(i,j) = 1 + \min\Big(\min_{1\leq k\leq i} \max\big(R_n(k-1,j),\ R_{n-k}(i-k,j)\big),$$
$$\min_{1\leq k\leq j} \max\big(R_n(i,k-1),\ R_{n-k}(i,j-k)\big)\Big)$$

при $0 \leq i \leq n$, $0 \leq j \leq n$, $i + j > 0$.

13. [*M42*] (Р. Л. Грэхем (R. L. Graham).) Покажите, что решение рекуррентного соотношения из упр. 12 можно выразить следующим образом. Определим функцию $G(x)$ при $0 < x < \infty$ такими правилами:

$$G(x) = \begin{cases} 1, & \text{если } 0 < x \leq \frac{5}{7}; \\ \frac{1}{2} + \frac{1}{8}G(8x-5), & \text{если } \frac{5}{7} < x \leq \frac{3}{4}; \\ \frac{1}{2}G(2x-1), & \text{если } \frac{3}{4} < x \leq 1; \\ 0, & \text{если } 1 < x < \infty. \end{cases}$$

(Рис. 38.) Поскольку $R_n(i,j) = R_n(j,i)$ и $R_n(0,j) = M(1,j)$, можно считать, что $1 \leq i \leq j \leq n$. Пусть $p = \lfloor \lg i \rfloor$, $q = \lfloor \lg j \rfloor$, $r = \lfloor \lg n \rfloor$ и пусть $t = n - 2^r + 1$. Тогда

$$R_n(i,j) = p + q + S_n(i,j) + T_n(i,j),$$

где функции S_n и T_n принимают значения 0 или 1:

$$S_n(i,j) = 1 \quad \text{тогда и только тогда, когда} \quad q < r \text{ или } (i - 2^p \geq u \text{ и } j - 2^r \geq u),$$
$$T_n(i,j) = 1 \quad \text{тогда и только тогда, когда} \quad p < r \text{ или } (t > \tfrac{6}{7}\, 2^{r-2} \text{ и } i - 2^r \geq v),$$

где $u = 2^p G(t/2^p)$ и $v = 2^{r-2} G(t/2^{r-2})$.

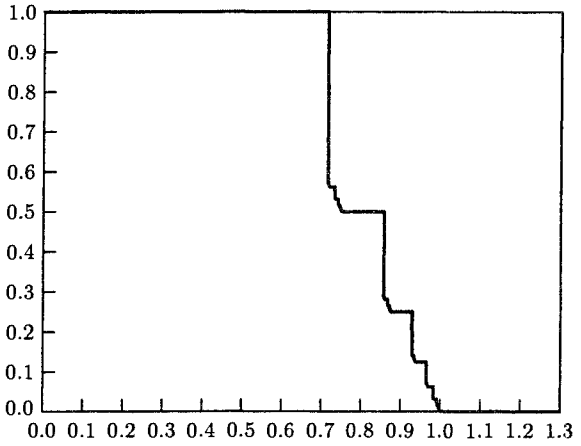


Рис. 38. Функция Грэхема (см. упр. 13).

(Это, быть может, самое сложное рекуррентное соотношение из всех, которые, возможно, когда-либо будут решены!)

14. [*41*] (Ф. К. Хуан.) Пусть $h_{3k} = \lfloor \frac{43}{28} 2^k \rfloor - 1$, $h_{3k+1} = h_{3k} + 3 \cdot 2^{k-3}$, $h_{3k+2} = \lfloor \frac{17}{7} 2^k - \frac{6}{7} \rfloor$ при $k \geq 3$ и начальные значения подобраны так, что

$$(h_0, h_1, h_2, \ldots) = (1, 1, 2, 2, 3, 4, 5, 7, 9, 11, 14, 18, 23, 29, 38, 48, 60, 76, \ldots).$$

Докажите, что $M(3, h_t) > t$ и $M(3, h_t - 1) \leq t$ при всех t, определяя таким образом точные значения $M(3, n)$ для всех n.

15. [*12*] На шаге H1 алгоритма бинарного слияния может потребоваться вычислить значение $\lfloor \lg(n/m) \rfloor$ при $n \geq m$. Как можно легко вычислить это значение, не применяя операций деления и взятия логарифма?

16. [*18*] При каких m и n, $1 \leq m \leq n \leq 10$, оптимален алгоритм бинарного слияния Хванга и Шень Линя?

17. [*M25*] Докажите неравенство (21). [*Указание.* Это неравенство не очень жесткое.]

18. [*M40*] Исследуйте *среднее* число сравнений, выполняемых алгоритмом бинарного слияния.

▶ **19.** [*23*] Докажите, что функция M удовлетворяет неравенству (22).

20. [*20*] Покажите, что если $M(m, n+1) \leq M(m+1, n)$ при всех $m \leq n$, то $M(m, n+1) \leq 1 + M(m, n)$ при всех $m \leq n$.

21. [*M47*] Докажите или опровергните соотношения (23) и (24).

22. [*M43*] Исследуйте минимальное *среднее* число сравнений, необходимых для слияния m элементов с n элементами.

23. [*M31*] (Э. Рейнгольд (E. Reingold).) Пусть $\{A_1, \ldots, A_n\}$ и $\{B_1, \ldots, B_n\}$ — множества, содержащие по n элементов. Рассмотрите алгоритм, который пытается проверить наличие *равенства* между множествами исключительно путем сравнений равенства элементов этих множеств. Таким образом, в процессе выполнения алгоритма задаются вопросы наподобие "$A_i = B_j$?" при некоторых i и j и выбирается дальнейший путь вычислений в зависимости от того, был ли ответ положительным или отрицательным.

Определив подходящего соперника, докажите, что любой такой алгоритм в наихудшем случае будет вынужден выполнить не менее $\frac{1}{2}n(n+1)$ сравнений.

24. [*22*] (Ю. Л. Лолер (E. L. Lawler).) Каково максимальное число сравнений, необходимых для следующего алгоритма слияния m элементов с $n \geq m$ элементами? "Установить $t \leftarrow \lfloor \lg(n/m) \rfloor$ и использовать алгоритм 5.2.4M для слияния $A_1, A_2, \ldots, A_m$ с $B_{2^t}, B_{2\cdot 2^t}, \ldots, B_{q\cdot 2^t}$, где $q = \lfloor n/2^t \rfloor$. Затем вставить каждый элемент A_j на свое место среди B_k."

▶ **25.** [*25*] Предположим, что (x_{ij}) есть матрица $m \times n$ с неубывающими строками и столбцами: $x_{ij} \leq x_{(i+1)j}$ при $1 \leq i < m$ и $x_{ij} \leq x_{i(j+1)}$ при $1 \leq j < n$. Покажите, что $M(m, n)$ есть минимальное число сравнений, необходимых для того, чтобы выяснить, присутствует ли данное число x в матрице, если все сравнения выполняются между x и некоторым элементом матрицы.

*5.3.3. Выбор с минимальным числом сравнений

При поиске наилучших возможных процедур для выбора t-го элемента в порядке убывания из n элементов мы встречаемся с классом задач, подобных рассмотренным в предыдущем разделе.

История постановки и поиска решений этой проблемы восходит к занимательному (хотя и серьезному) эссе преподобного Ч. Л. Доджсона (C. L. Dodgson) о турнирах по теннису, появившемуся в *St. James's Gazette* 1 августа 1883 года (с. 5–6). Доджсон, который, разумеется, более известен как Льюис Кэррол, рассматривал несправедливые правила, по которым присуждались (и до сих пор присуждаются) призы в турнирах по теннису. Рассмотрим, например, рис. 39, на котором показана схема типичного турнира с выбыванием между 32 игроками, помеченными *01*, *02*, ..., *32*. В финале игрок *01* одерживает победу над игроком *05*, поэтому ясно, что игрок *01* — чемпион и заслужил первый приз. Несправедливость проявляется в том, что игрок *05* обычно получает второй приз, хотя он может и не быть вторым по силе игроком. Выиграть второй приз можно, даже если играешь хуже половины игроков турнира. В самом деле, как заметил Доджсон, второй по силе игрок выигрывает второй приз в том и только том случае, если первоначально он и чемпион находились в противоположных частях турнирной таблицы; для турнира с 2^n участниками это происходит с вероятностью $2^{n-1}/(2^n - 1)$, так что почти в половине случаев второй приз получает не тот игрок! Если проигравшие в полуфинале (игроки *25* и *17* на рис. 39) соревнуются за третий приз, то весьма маловероятно, что третий по силе игрок получит третий приз.

Поэтому Доджсон решил найти такую схему турнира, которая правильно определяет второго и третьего по силе игроков в предположении, что соблюдается закон транзитивности. (Иначе говоря, если игрок A побеждает игрока B, а B побеждает C, то можно считать, что игрок A победит C.) Доджсон придумал процедуру, в которой проигравшим дается возможность сыграть еще несколько игр, пока не станет определенно известно, что они слабее других трех игроков. Пример схемы Доджсона приводится на рис. 40. Здесь изображена сетка игр дополнительного турнира; его следует провести вместе с турниром, схема которого показана на рис. 39. Доджсон попытался составить пары игроков, у которых до сих пор были равные результаты, и исключить матчи между игроками, побежденными одним и тем же участником. В таком конкретном примере игрок *16* проигрывает игроку *11*, а игрок *13* проигрывает игроку *12* в первом туре; после того как игрок *16* проигрывает игроку *13* во втором туре, игрок *16* исключается, так как теперь известно, что он слабее игроков *11*, *12* и *13*. В третьем туре исключается встреча игроков *19* и *21*,

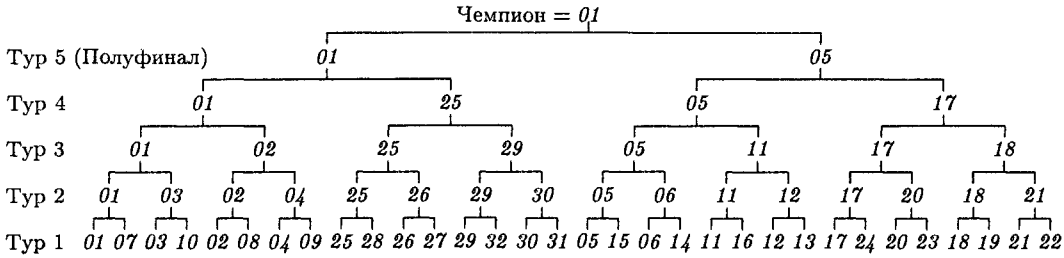


Рис. 39. Турнир между 32 игроками с выбыванием.

так как они оба были побеждены игроком *18*, но нельзя автоматически исключить проигравшего во встрече игроков *19* и *21*.

Было бы приятно сообщить, что схема турнира, предложенная Льюисом Кэрролом, оказалась оптимальной, но, к сожалению, это не так. Из записи в его дневнике от 23 июля 1883 года явствует, что он составил это эссе примерно за шесть часов и чувствовал, что, "поскольку [теннисный] сезон приближается к концу, будет лучше сделать так, чтобы оно [эссе] появилось побыстрее, чем чтобы оно было хорошо написано". В его процедуре делается больше сравнений, чем необходимо, и она не сформулирована достаточно четко, чтобы квалифицировать ее как алгоритм. С другой стороны, в ней имеются некоторые очень интересные аспекты, если судить с точки зрения параллельных вычислений. Предложенную схему можно считать отличной турнирной сеткой, поскольку Кэррол включил в нее несколько драматических эффектов; например, он определил, что два финалиста должны пропустить пятый тур и сыграть дополнительный матч в турах 6 и 7. Однако организаторы турниров, по-видимому, сочли его предложение излишне логичным, и потому система Кэррола, скорее всего, никогда не испытывалась. Вместо этого практикуется метод "рассеивания" более сильных игроков, чтобы они попали в разные части дерева.

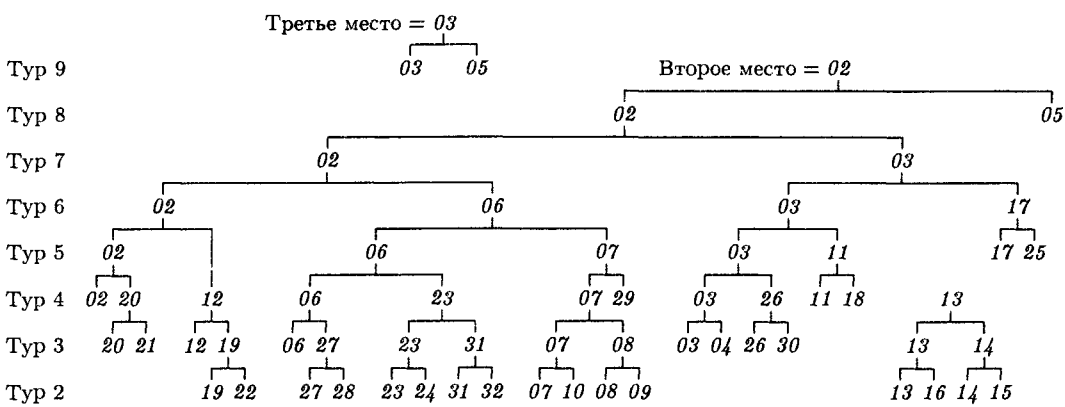


Рис. 40. Сетка теннисного турнира Льюиса Кэррола (в дополнение к сетке, представленной на рис. 39).

На математическом семинаре в 1929–1930 годах Гуго Штейнгауз (Hugo Steinhaus) сформулировал постановку задачи нахождения *минимального* числа поединков, требуемых для определения первого и второго игроков в турнире, если всего

имеется $n \geq 2$ участников. В работе J. Schreier, *Mathesis Polska* **7** (1932), 154–160, приведена процедура, требующая самое большее $n - 2 + \lceil \lg n \rceil$ матчей. В ней использован, по существу, тот же метод, что и на первых двух стадиях процесса, который мы назвали сортировкой посредством выбора из дерева (см. раздел 5.2.3, рис. 23), однако не выполняли дополнительных сравнений с $-\infty$. В работе также утверждается, что $n - 2 + \lceil \lg n \rceil$ — наилучшее возможное значение, но приведенное доказательство было ошибочным, как и еще одна попытка доказательства, предпринятая в работе J. Słupecki, *Colloquium Mathematicum* **2** (1951), 286–290. Прошло 32 года, прежде чем С. С. Кислицыным было опубликовано правильное, хотя и очень сложное, доказательство [*Сибирский математический журнал* **5** (1964), 557–564].

Пусть $V_t(n)$ обозначает минимальное число сравнений, требуемых для определения t-го в порядке убывания элемента из n элементов, $1 \leq t \leq n$, и пусть $W_t(n)$ равно наименьшему числу сравнений, необходимых для определения наибольшего, второго, ... и t-го элементов всех сразу. Вследствие симметрии имеем

$$V_t(n) = V_{n+1-t}(n), \tag{1}$$

очевидно также, что

$$V_1(n) = W_1(n), \tag{2}$$

$$V_t(n) \leq W_t(n), \tag{3}$$

$$W_n(n) = W_{n-1}(n) = S(n). \tag{4}$$

Как следует из леммы 5.2.3M,

$$V_1(n) = n - 1. \tag{5}$$

Существует удивительно простое доказательство этого факта — каждый участник турнира, кроме чемпиона, должен проиграть, по крайней мере, одну игру! Обобщая эту идею и используя "соперника", как в разделе 5.3.2, можно без особого труда доказать теорему Шрейера-Кислицына.

Теорема S. $V_2(n) = W_2(n) = n - 2 + \lceil \lg n \rceil$ при $n \geq 2$.

Доказательство. Предположим, что в турнире, в котором после проведения некоторой заданной процедуры должен определиться второй игрок, участвуют n игроков и пусть a_j — число игроков, проигравших j или больше матчей. Общее число сыгранных матчей будет тогда равно $a_1 + a_2 + a_3 + \cdots$. Нельзя определить второго игрока, не выявив заодно и чемпиона (см. упр. 2), поэтому из предыдущих рассуждений получаем $a_1 = n - 1$. Для завершения доказательства покажем, что всегда существует последовательность результатов матчей, которая приводит к $a_2 \geq \lceil \lg n \rceil - 1$. Предположим, что к концу турнира чемпион сыграл p игр и победил p игроков; одним из них был второй по силе игрок, а остальные должны проиграть, по крайней мере, еще по одному разу, поэтому $a_2 \geq p - 1$. Итак, можно завершить доказательство, построив соперника, предопределяющего результаты игр, таким образом, чтобы чемпиону пришлось сыграть еще хотя бы с $\lceil \lg n \rceil$ другими участниками турнира.

Пусть соперник считает, что игрок A сильнее игрока B, если A ранее не проигрывал, а B хотя бы однажды проиграл или если оба не проигрывали, но B выиграл к этому моменту меньше поединков, чем A. При других обстоятельствах

соперник может принимать произвольное решение, не противоречащее некоторому частичному упорядочению.

Рассмотрим результаты завершенного турнира, матчи которого предопределялись таким соперником. Мы скажем, что "A превосходит B" тогда и только тогда, когда $A = B$ или A превосходит игрока, который первым победил B. (Только первое поражение игрока существенно в этом отношении, последующие его игры игнорируются. В соответствии с поведением соперника любой игрок, *первым* победивший какого-либо другого игрока, ни в одной из предыдущих встреч не должен иметь поражений.) Отсюда следует, что игрок, который выиграл свои первые p матчей, превосходит на основании этих p игр не более 2^p игроков. (Если $p = 0$, это очевидно, если же $p > 0$, то p-й матч был сыгран против игрока, который либо ранее потерпел поражение, либо превосходит не более 2^{p-1} игроков.) Чемпион превосходит всех, поэтому он должен был сыграть не менее $\lceil \lg n \rceil$ матчей. ▮

Таким образом, задача нахождения второго в порядке убывания элемента полностью решена с помощью теоремы S в "минимаксном" смысле. В упр. 6 показано, что можно дать простую формулу для минимального числа сравнений, необходимых для выявления второго элемента множества, если известно *произвольное* частичное упорядочение элементов.

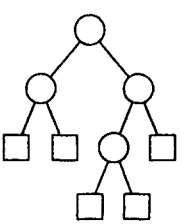
А что будет, если $t > 2$? В упомянутой статье Кислицын пошел дальше. Он рассмотрел большие значения t, доказав, что

$$W_t(n) \le n - t + \sum_{n+1-t<j\le n} \lceil \lg j \rceil \qquad \text{при } n \ge t. \tag{6}$$

Мы видели, что при $t = 1$ и $t = 2$ эта формула представляет собой равенство; при $t = 3$ она может быть слегка улучшена (см. упр. 21).

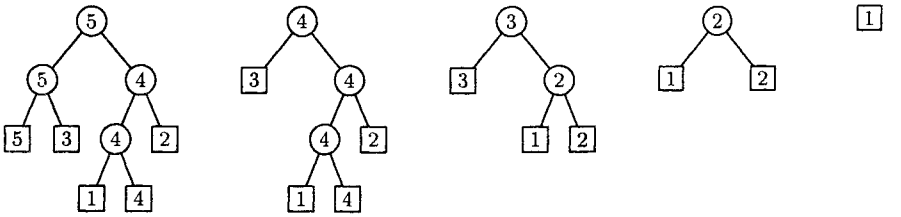
Мы докажем теорему Кислицына, показав, что первые t стадий *выбора из дерева* требуют не более $n - t + \sum_{n+1-t<j\le n} \lceil \lg j \rceil$ сравнений (исключая все сравнения с $-\infty$). Интересно, что правая часть (6) равна $B(n)$, когда $t = n$, а также когда $t = n - 1$ [см. формулу 5.3.1–(3)]; следовательно, методы выбора из дерева и бинарных вставок приводят к одной и той же верхней оценке для задачи сортировки, хотя это совершенно различные методы.

Пусть α — расширенное бинарное дерево с n внешними узлами, а π — перестановка множества $\{1, 2, \ldots, n\}$. Поместим элементы перестановки π во внешние узлы слева направо в симметричном порядке и заполним внутренние узлы в соответствии с правилами турнира с выбыванием, как при выборе из дерева. Повторно применив операцию выбора к результирующему дереву, можно определить последовательность $c_{n-1}\, c_{n-2} \ldots c_1$, где c_j есть число сравнений, требуемых, чтобы перенести элемент j в корень дерева после того, как элемент $j + 1$ будет заменен элементом $-\infty$. Например, если α — дерево



(7)

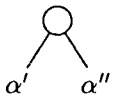
и если $\pi = 5\ 3\ 1\ 4\ 2$, то мы получаем последовательные деревья



$c_4 = 1$ $\quad$ $c_3 = 2$ $\quad$ $c_2 = 0$ $\quad$ $c_1 = 0$

Если же $\pi = 3\ 1\ 5\ 4\ 2$, то последовательность $c_4\, c_3\, c_2\, c_1$ будет иной, а именно — 2 1 1 0. Легко видеть, что c_1 всегда есть 0.

Пусть $\mu(\alpha, \pi)$ — мультимножество $\{c_{n-1}, c_{n-2}, \ldots, c_1\}$, определяемое α и π. Если

$\alpha =$ 

и если элементы 1 и 2 не содержатся оба либо в α', либо в α'', то легко видеть, что

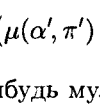
$$\mu(\alpha, \pi) = \big(\mu(\alpha', \pi') + 1\big) \uplus \big(\mu(\alpha'', \pi'') + 1\big) \uplus \{0\} \tag{8}$$

для подходящих перестановок π' и π'', где $\mu + 1$ обозначает мультимножество, получаемое в результате прибавления 1 к каждому элементу μ (см. упр. 7). С другой стороны, если и элемент 1, и элемент 2 находятся в α', имеем

$$\mu(\alpha, \pi) = \big(\mu(\alpha', \pi') + \epsilon\big) \uplus \big(\mu(\alpha'', \pi'') + 1\big) \uplus \{0\},$$

где $\mu + \epsilon$ обозначает какое-нибудь мультимножество, получаемое в результате прибавления 1 к некоторым элементам μ и 0 — к остальным. Аналогичная формула справедлива и в том случае, если элементы 1 и 2 находятся в α''. Будем говорить, что мультимножество μ_1 *мажорирует* μ_2, если и μ_1, и μ_2 содержат равное число элементов и если k-й в порядке убывания элемент μ_1 больше k-го в порядке убывания элемента μ_2 для всех k или равен ему. Определим $\mu(\alpha)$ как мажоранту $\mu(\alpha, \pi)$ по всем перестановкам π в том смысле, что $\mu(\alpha)$ мажорирует $\mu(\alpha, \pi)$ при всех π и $\mu(\alpha) = \mu(\alpha, \pi)$ при некотором π. Приведенные выше формулы показывают, что

$$\mu(\square) = \emptyset, \qquad \mu(\bigcirc) = \big(\mu(\alpha') + 1\big) \uplus \big(\mu(\alpha'') + 1\big) \uplus \{0\}; \tag{9}$$



следовательно, $\mu(\alpha)$ *есть мультимножество всех расстояний от корня до внутренних узлов* α.

Если читатель уследил за ходом наших рассуждений, ему должно быть ясно, что теперь мы готовы доказать теорему Кислицына (6). Поскольку $W_t(n)$ меньше или равно $n - 1$ плюс $t - 1$ наибольших элементов $\mu(\alpha)$, где α — любое дерево, используемое при сортировке посредством выбора из дерева, то можно считать α полным бинарным деревом с n внешними узлами (см. раздел 2.3.4.5), причем

$$\mu(\alpha) = \{\lfloor \lg 1 \rfloor, \lfloor \lg 2 \rfloor, \ldots, \lfloor \lg(n-1) \rfloor\}$$
$$= \{\lceil \lg 2 \rceil - 1, \lceil \lg 3 \rceil - 1, \ldots, \lceil \lg n \rceil - 1\}. \tag{10}$$

Мы получим формулу (6), если рассмотрим $t-1$ наибольших элементов этого мультимножества.

Теорема Кислицына дает хорошую верхнюю оценку для $W_t(n)$; Кислицын отметил, что $V_3(5) = 6 < W_3(5) = 7$, но не смог найти в общем случае оценку для $V_t(n)$, лучшую, чем для $W_t(n)$. Это было сделано А. Хадьяном (A. Hadian) и М. Собелем (M. Sobel), которые использовали *выбор с замещением* вместо выбора из дерева (см. раздел 5.4.1). Выведенная ими формула [Univ. of Minnesota, Dept. of Statistics Report 121 (1969)],

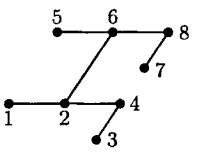
$$V_t(n) \le n - t + (t-1)\lceil \lg(n+2-t) \rceil, \qquad n \ge t, \tag{11}$$

отличается от формулы Кислицына тем, что каждый элемент суммы в (6) заменен наименьшим элементом.

Теорему Хадьяна и Собеля (11) можно доказать, воспользовавшись следующей схемой. Сначала сформируем бинарное дерево для турнира с выбыванием $n-t+2$ элементов (участников), что потребует $n-t+1$ сравнений. Наибольший элемент превосходит $n-t+1$ других элементов, поэтому он не может быть t-м в порядке убывания. Заменим его во внешнем узле дерева одним из $t-2$ элементов, оставшихся в резерве, и найдем наибольший элемент из образовавшегося набора $n-t+2$ элементов; это требует не более $\lceil \lg(n+2-t) \rceil$ сравнений, поскольку придется заново вычислить только один путь в дереве. Повторим эту операцию в общей сложности $t-2$ раз — по одному разу для каждого элемента из резерва. Наконец, заменим текущий наибольший элемент элементом $-\infty$ и определим наибольший из оставшихся $n+1-t$ элементов; для этого потребуется не более $\lceil \lg(n+2-t) \rceil - 1$ сравнений, и t-й в порядке убывания элемент исходного множества попадет в корень дерева. Суммировав число сравнений, получаем выражение (11).

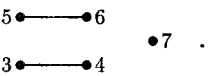
Разумеется, мы должны заменить t на $n+1-t$ в правой части соотношения (11), если $n+1-t$ дает лучшее значение (как при $n=6$ и $t=3$). Как ни странно, но эта формула дает для $V_7(13)$ меньшую оценку, чем для $V_6(13)$. Верхняя оценка в (11) точна для $n \le 6$, но когда n и t становятся бо́льшими, можно получить значительно лучшие оценки для $V_t(n)$.

Например, можно использовать следующий изящный метод (принадлежащий Дэвиду Г. Дорену (David G. Doren)), чтобы показать, что $V_4(8) \le 12$. Обозначим элементы через $X_1, \ldots, X_8$. Сначала сравним $X_1 : X_2$ и $X_3 : X_4$, а затем — двух "победителей" между собой; проделаем то же самое для пар $X_5 : X_6$ и $X_7 : X_8$ и "победителей" этих пар. Переименуем элементы так, чтобы получить $X_1 < X_2 < X_4 > X_3$, $X_5 < X_6 < X_8 > X_7$, затем сравним $X_2 : X_6$; в силу симметрии положим $X_2 < X_6$, поэтому получим конфигурацию



.

(Теперь X_1 и X_8 вышли из игры и мы должны найти третий в порядке убывания элемент из $\{X_2, \ldots, X_7\}$.) Сравним $X_2 : X_7$ и отбросим меньший; в худшем случае получим $X_2 < X_7$. Найдем третий в порядке убывания элемент из



Это можно сделать еще за $V_3(5) - 2 = 4$ шага, так как процедура для $V_3(5) = 6$ в (11) начинается со сравнения двух непересекающихся пар элементов.

Можно использовать другие трюки подобного вида и получить результаты, показанные в табл. 1, но никакого общего метода до сих пор не существует. Значения, указанные для $V_4(9) = V_6(9)$ и $V_5(10) = V_6(10)$, являются оптимальными, как доказано в работе W. Gasarch, W. Kelly, W. Pugh, *SIGACT News* **27**, 2 (June, 1996), 88–96, в которой использовался компьютерный поиск.

Хорошая нижняя оценка для задачи выбора в случае, когда t мало́, получена в работе Дэвида Г. Киркпатрика (David G. Kirkpatrick) [*JACM* **28** (1981), 150–165]: если $2 \le t \le (n+1)/2$, получим

$$V_t(n) \ge n + t - 3 + \sum_{j=0}^{t-2} \left\lceil \lg \frac{n-t+2}{t+j} \right\rceil . \tag{12}$$

В своей диссертации [Ph. D. thesis, U. of Toronto, 1974] он доказал, что

$$V_3(n) \le n + 1 + \left\lceil \lg \frac{n-1}{4} \right\rceil + \left\lceil \lg \frac{n-1}{5} \right\rceil ; \tag{13}$$

эта нижняя оценка соответствует нижней оценке, что следует из (12) при $\lg \frac{5}{3} \approx 74\%$ всех целых чисел n и превосходит (12) не более чем на 1. Выполненный Киркпатриком анализ дает основание предположить, что равенство в (13) будет соблюдаться и для всех $n > 4$, но Ютта Эстерброк (Jutta Eusterbrock) отыскала удивительный противоречащий этому пример $V_3(22) = 28$ [*Discrete Applied Math.* **41** (1993), 131–137]. Несколько улучшенная нижняя оценка для больших значений t найдена С. У. Бентом (S. W. Bent) и Дж. У. Джоном (J. W. John) (см. упр. 26):

$$V_t(n) \ge n + m - 2\lceil \sqrt{m} \rceil, \qquad m = 2 + \left\lceil \lg \left(\binom{n}{t} \Big/ (n+1-t) \right) \right\rceil . \tag{14}$$

Эта формула доказывает, в частности, что

$$V_{\alpha n}(n) \ge \left(1 + \alpha \lg \frac{1}{\alpha} + (1-\alpha) \lg \frac{1}{1-\alpha}\right) n + O(\sqrt{n}). \tag{15}$$

Линейный метод. Если n нечетно и $t = \lceil n/2 \rceil$, то t-й в порядке убывания (и t-й в порядке возрастания) элемент называется медианой. В соответствии с (11) мы можем найти медиану n элементов за $\approx \frac{1}{2} n \lg n$ сравнений, но это лишь приблизительно вдвое быстрее сортировки, хотя в таком случае нам нужно значительно меньше информации. В течение нескольких лет объединенные усилия множества исследователей были направлены на улучшение формулы (11) для больших значений t и n; наконец, в 1971 году Мануэль Блум (Manuel Blum) открыл метод, требующий только $O(n \log \log n)$ шагов. Подход Блума к этой задаче дал толчок к

Таблица 1

ОЦЕНКИ $V_t(n)$ ПРИ МАЛЫХ n

n	$V_1(n)$	$V_2(n)$	$V_3(n)$	$V_4(n)$	$V_5(n)$	$V_6(n)$	$V_7(n)$	$V_8(n)$	$V_9(n)$	$V_{10}(n)$
1	0									
2	1	1								
3	2	3	2							
4	3	4	4	3						
5	4	6	6	6	4					
6	5	7	8	8	7	5				
7	6	8	10	10*	10	8	6			
8	7	9	11	12	12	11	9	7		
9	8	11	12	14	14*	14	12	11	8	
10	9	12	14*	15	16**	16**	15	14*	12	9

* Для этих случаев в упр. 10–12 приводятся схемы, позволяющие улучшить (11).
** См. K. Noshita, *Trans. of the IECE of Japan* **E59**, 12 (December, 1976), 17–18.

развитию нового класса методов, который привел к следующему построению (см. R. Rivest, R. Tarjan, *J. Comp. and Sys. Sci.* **7** (1973), 448–461).

Теорема L. $V_t(n) \le 15n - 163$ *при* $1 \le t \le n$, *когда* $n > 32$.

Доказательство. Когда n мало́, теорема тривиальна, так как $V_t(n) \le S(n) \le 10n \le 15n - 163$ для $32 < n \le 2^{10}$. Добавив самое большее 13 фиктивных элементов $-\infty$, можно считать, что $n = 7(2q+1)$ при некотором целом $q \ge 73$. Теперь для выбора t-го в порядке убывания элемента воспользуемся следующим методом.

Шаг 1. Разобьем элементы на $2q+1$ групп по 7 элементов и отсортируем каждую группу. Это потребует не более $13(2q+1)$ сравнений.

Шаг 2. Найдем медиану из $2q+1$ медиан, полученных на шаге 1, и обозначим ее через x. Проведя индукцию по q, замечаем, что это требует не более $V_{q+1}(2q+1) \le 30q - 148$ сравнений.

Шаг 3. Теперь $n-1$ элементов, отличных от x, разбиваются на три множества (рис. 41):

$4q+3$ элементов, о которых известно, что они больше x (область B);

$4q+3$ элементов, о которых известно, что они меньше x (область C);

$6q$ элементов, отношение которых к x неизвестно (области A, D).

Выполнив дополнительно $4q$ сравнений, можно в точности сказать, какие элементы из областей A и D меньше x. (Сначала сравниваем x со средним элементом каждой тройки.)

Шаг 4. Теперь при некотором r мы нашли r элементов, больших, чем x, а также x и $n-1-r$ элементов, меньших, чем x. Если $t = r+1$, то x и будет ответом; если $t < r+1$, то нужно найти t-й элемент в порядке убывания из r больших элементов; если $t > r+1$, то нужно найти $(t-1-r)$-й элемент в порядке убывания из $n-1-r$ меньших элементов. Суть дела в том, что и r, и $n-1-r$ меньше или равны $10q+3$ (размер областей A и D плюс B или C). Индукцией по q выводим, что этот шаг требует не более $15(10q+3) - 163$ сравнений.

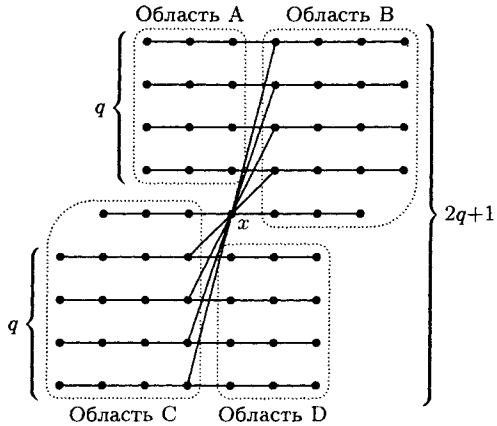


Рис. 41. Алгоритм выбора Райвеста и Таржана ($q = 4$).

Общее число сравнений оказывается не больше

$$13(2q+1) + 30q - 148 + 4q + 15(10q+3) - 163 = 15(14q-6) - 163.$$

Так как мы начали с не менее $14q - 6$ элементов, доказательство завершено. ▌

Из теоремы L следует, что время выбора всегда может быть линейным, а именно — что $V_t(n) = O(n)$. Метод, использованный в этом доказательстве, не вполне совершенный, поскольку на шаге 4 теряется значительное количество информации. Более глубокий анализ задачи приводит к уточнению оценок границ. (См., например, работу A. Schönhage, M. Paterson, N. Pippenger, *J. Comp. Sys. Sci.* **13** (1976), 184–199, в которой доказано, что максимальное число сравнений, необходимых для поиска медианы, не превосходит $3n + O(n \log n)^{3/4}$. По поводу нижней оценки числа сравнений обратитесь к упр. 23; в нем же приведены ссылки на более поздние источники.)

Среднее число. Вместо минимизации *максимального* числа сравнений можно искать алгоритм, который минимизирует *среднее* число сравнений, предполагая, что порядок случаен. Как обычно, эта задача значительно труднее и она все еще не решена даже в случае $t = 2$. Клод Пикар (Claude Picard) упомянул о ней в своей книге *Théorie des Questionnaires* (1965). Широкое исследование было предпринято Милтоном Собелем (Milton Sobel) [Univ. of Minnesota, Dept. of Statistics, report 113 (November, 1968)]; *Revue Française d'Automatique, Informatique et Recherche Opérationnelle* **6**, R-3 (December, 1972), 23–68].

Собель построил процедуру, графически представленную на рис. 42, которая находит второй в порядке убывания элемент из шести элементов, в среднем используя только $6\frac{1}{2}$ сравнений. В худшем случае требуется восемь сравнений, и это хуже, чем $V_2(6) = 7$. Предпринятый Д. Хоэем (D. Hoey) продолжительный компьютерный эксперимент показал, что в наилучшей процедуре решения этой задачи используется в среднем $6\frac{26}{45}$ сравнений, если ограничить эксперимент семью сравнениями. Таким образом, вероятно, никакая процедура нахождения второго из шести элементов не будет оптимальной одновременно и как минимаксная, и как минимизирующая среднее число сравнений.

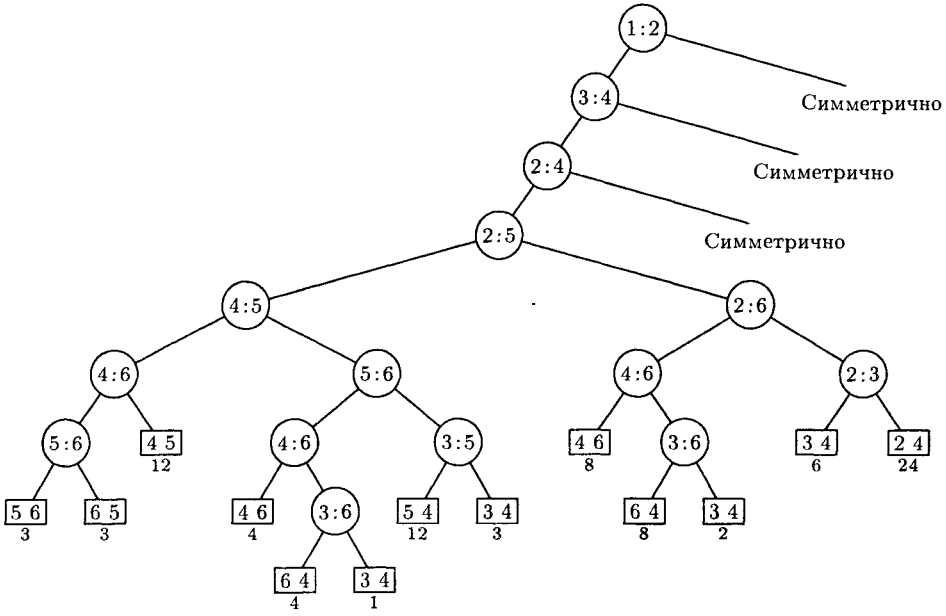


Рис. 42. Процедура выбора второго в порядке убывания элемента из $\{X_1, X_2, X_3, X_4, X_5, X_6\}$, при которой используется в среднем $6\frac{1}{2}$ сравнений. Каждая "симметричная" ветвь идентична своему собрату, однако имена переставлены соответствующим образом. Во внешних узлах записано "j k", если известно, что X_j — второй, а X_k — наибольший элемент; число перестановок, приводящих к этому узлу, записано непосредственно под ним.

Пусть $\overline{V}_t(n)$ — минимальное среднее число сравнений, необходимых для определения t-го элемента в порядке убывания из n элементов. В табл. 2 показаны точные значения для малых n, вычисленные Д. Хоэем.

Р. У. Флойд (R. W. Floyd) в 1970 году обнаружил, что для поиска медианы n элементов может потребоваться в среднем всего $\frac{3}{2}n + O(n^{2/3}\log n)$ сравнений. Хе (He) и Р. Л. Ривест (R. L. Rivest) спустя несколько лет уточнили этот результат и сформулировали изящный алгоритм доказательства того, что

$$\overline{V}_t(n) \le n + \min(t, n-t) + O\left(\sqrt{n\log n}\right). \tag{16}$$

(См. упр. 13 и 24.)

Используя несколько отличный подход, который основан на обобщении построения Собеля при $t = 2$, Дэвид У. Матула (David W. Matula) [Washington Univ. Tech. Report AMCS-73-9 (1973)] показал, что

$$\overline{V}_t(n) \le n + t\lceil\lg t\rceil(11 + \ln\ln n). \tag{17}$$

Таким образом, для фиксированного t средний объем вычислений может быть снижен до $n + O(\log\log n)$ сравнений. Изящная нижняя оценка $\overline{V}_t(n)$ представлена в упр. 25.

Задачи сортировки и поиска представляют собой частные случаи более общей задачи поиска перестановки n заданных элементов, которая имеет заданное частич-

Таблица 2

МИНИМАЛЬНОЕ СРЕДНЕЕ ЧИСЛО СРАВНЕНИЙ ПРИ ВЫБОРЕ

n	$\overline{V}_1(n)$	$\overline{V}_2(n)$	$\overline{V}_3(n)$	$\overline{V}_4(n)$	$\overline{V}_5(n)$	$\overline{V}_6(n)$	$\overline{V}_7(n)$
1	0						
2	1	1					
3	2	$2\frac{2}{3}$	2				
4	3	4	4	3			
5	4	$5\frac{4}{15}$	$5\frac{13}{15}$	$5\frac{4}{15}$	4		
6	5	$6\frac{1}{2}$	$7\frac{7}{18}$	$7\frac{7}{18}$	$6\frac{1}{2}$	5	
7	6	$7\frac{149}{210}$	$8\frac{509}{630}$	$9\frac{32}{105}$	$8\frac{509}{630}$	$7\frac{149}{210}$	6

ное упорядочение. В работе A. C. Yao, *SICOMP* **18** (1989), 679–689, показано, что если частичное упорядочение задано ациклическим диграфом G на n вершинах с k связанными компонентами, то минимальное число сравнений, необходимых для разрешения проблемы, всегда равно $\Theta\big(\lg(n!/T(G))+n-k\big)$, как в худшем случае, так и в среднем, где $T(G)$ — суммарное число частично упорядоченных составляющих в перестановках (число топологических сортировок в G).

УПРАЖНЕНИЯ

1. [*15*] Почему в турнире Льюиса Кэррола (см. рис. 39 и 40) игрок *13* выбывает, несмотря на то что он выиграл свой матч в третьем туре?

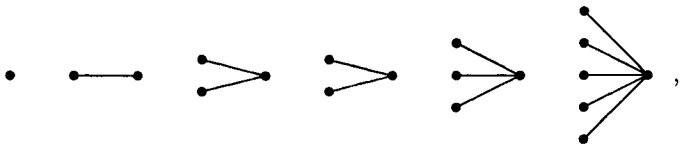
▶ **2.** [*M25*] Докажите, что после того, как найден с помощью последовательности сравнений t-й элемент в порядке убывания из n элементов, также известно, какие $t-1$ элементов больше него и какие $n-t$ элементов меньше.

3. [*20*] Докажите, что $V_t(n) > V_t(n-1)$ и $W_t(n) > W_t(n-1)$ при $1 \le t < n$.

▶ **4.** [*M25*] (Ф. Фюснегер (F. Fussenegger) и Г. Н. Габов (H. N. Gabow).) Докажите, что $W_t(n) \ge n - t + \lceil \lg n^{\underline{t-1}} \rceil$.

5. [*10*] Докажите, что $W_3(n) \le V_3(n) + 1$.

▶ **6.** [*M26*] (Р. У. Флойд.) Дано n различных элементов $\{X_1, \ldots, X_n\}$ и набор отношений $X_i < X_j$ для некоторых пар (i, j). Нужно найти второй в порядке убывания элемент. Элемент X_i не может быть вторым, если известно, что $X_i < X_j$ и $X_i < X_k$ при $j \ne k$, поэтому его можно исключить. В результате отношения будут иметь вид



а именно — образуется m групп элементов, которые можно представить мультимножеством $\{l_1, l_2, \ldots, l_m\}$; j-я группа содержит $l_j + 1$ элементов, об одном из которых известно, что он больше остальных. Например, изображенная конфигурация может быть описана мультимножеством $\{0, 1, 2, 2, 3, 5\}$; если ни одно отношение не известно, то имеем вектор из n нулей.

Пусть $f(l_1, l_2, \ldots, l_m)$ — минимальное число сравнений, необходимых для определения второго элемента такого частично упорядоченного множества. Докажите, что

$$f(l_1, l_2, \ldots, l_m) = m - 2 + \lceil \lg(2^{l_1} + 2^{l_2} + \cdots + 2^{l_m}) \rceil.$$

[*Указание.* Покажите, что наилучшая стратегия всегда состоит в том, чтобы сравнивать наибольшие элементы двух самых маленьких групп, пока m не будут сведены к единице; используйте индукцию по $l_1 + l_2 + \cdots + l_m + 2m$.]

7. [*M20*] Докажите (8).

8. [*M21*] Формула Кислицына (6) основана на сортировке посредством выбора из дерева, использующей полное бинарное дерево с n внешними узлами. Может ли выбор из дерева, основанный на некотором *другом* дереве, дать лучшую оценку для каких-нибудь t и n?

▶ **9.** [*20*] Нарисуйте дерево сравнений, чтобы найти медиану пяти элементов не более чем за шесть шагов, используя метод выбора с замещением Хадьяна и Собеля [см. (11)].

10. [*35*] Покажите, что медиана семи элементов может быть найдена не более чем за десять шагов.

11. [*38*] (К. Ношита (K. Noshita).) Покажите, что медиана девяти элементов может быть найдена не более чем за 14 шагов, из которых первые семь идентичны методу Дорена.

12. [*21*] (Хадьян (A. Hadian) и Собель (M. Sobel).) Докажите, что $V_3(n) \le V_3(n-1) + 2$. [*Указание.* Начните с удаления наименьшего из $\{X_1, X_2, X_3, X_4\}$.]

▶ **13.** [*HM28*] (Р. У. Флойд.) Покажите, что если начать с нахождения медианы $\{X_1, \ldots, X_{n^{2/3}}\}$ с помощью рекурсивно определенного метода, то можно найти медиану $\{X_1, \ldots, X_n\}$, выполнив в среднем $\frac{3}{2}n + O(n^{2/3} \log n)$ сравнений.

▶ **14.** [*20*] (М. Собель (M. Sobel)).) Пусть $U_t(n)$ — минимальное число сравнений, необходимых для поиска t наибольших из n элементов, если не важен их взаимный порядок. Покажите, что $U_2(5) \le 5$.

15. [*22*] (А. Пол (I. Pohl).) Предположим, что нас интересует минимизация объема памяти, а не времени. Какое минимальное количество слов памяти требуется для вычисления t-го из n элементов, если каждый элемент занимает одно слово и элементы вводятся в особый регистр по одному?

▶ **16.** [*25*] (И. Пол.) Покажите, что можно найти одновременно максимум и минимум множества из n элементов, использовав не более $\lceil \frac{3}{2}n \rceil - 2$ сравнений, и это число не может быть уменьшено. [*Указание.* Любая стадия такого алгоритма может быть представлена четверкой (a, b, c, d), где a элементов вообще не сравнивались, b элементов выигрывали, но никогда не проигрывали, c проигрывали, но никогда не выигрывали, d как выигрывали, так и проигрывали. Постройте подходящего соперника.]

17. [*20*] (Р. У. Флойд.) Покажите, что можно выбрать k наибольших и l наименьших элементов множества из n элементов, использовав не более $\lceil \frac{3}{2}n \rceil - k - l + \sum_{n+1-k<j\le n} \lceil \lg j \rceil + \sum_{n+1-l<j\le n} \lceil \lg j \rceil$ сравнений.

18. [*M20*] Если бы в доказательстве теоремы L использовались группы размером 5, а не 7, то какая бы получилась теорема?

19. [*M42*] Расширьте табл. 2 для $n = 8$.

20. [*M47*] Каково значение асимптотического выражения для $\bar{V}_2(n) - n$ при $n \to \infty$?

21. [*32*] (П. В. Раманан (P. V. Ramanan) и Л. Хьяфил (L. Hyafil).) Докажите, что $W_t(2^k + 2^{k+1-t}) \le 2^k + 2^{k+1-t} + (t-1)(k-1)$, если $k \ge t \ge 2$; покажите также, что имеется равенство для бесконечно больших k и t, используя результаты упр. 4. [*Указание.* Постройте два дерева турниров с выбыванием и разумно скомбинируйте полученные результаты.]

22. [*24*] (Дэвид Г. Киркпатрик (David G. Kirkpatrick).) Покажите, что в случае $4 \cdot 2^k < n-1 \le 5 \cdot 2^k$ верхняя оценка (11) для $V_3(n)$ может быть следующим образом уменьшена на 1. (i) Образуйте четыре "дерева с выбыванием" размером 2^k. (ii) Найдите минимальный из четырех максимумов и удалите все 2^k элементов соответствующего дерева. (iii) Используя

накопленную информацию, постройте одно дерево с выбыванием размером $n - 1 - 2^k$. (iv) Продолжайте, как при доказательстве (11).

23. [*M49*] Каково асимптотическое значение $V_{\lceil n/2 \rceil}(n)$ при $n \to \infty$?

24. [*HM40*] Докажите, что $\overline{V}_t(n) \le n + t + O(\sqrt{n \log n})$ при $t \le \lceil n/2 \rceil$. *Указание.* Покажите, что, используя столько сравнений, можно фактически найти и $\lfloor t - \sqrt{t \ln n} \rfloor$-й, и $\lceil t + \sqrt{t \ln n} \rceil$-й элементы, после чего легко выявляется t-й.

▸ **25.** [*M35*] (В. Кунто (W. Cunto) и Дж. И. Мунро (J. I. Munro).) Докажите, что $\overline{V}_t(n) \ge n + t - 2$, если $t \le \lceil n/2 \rceil$.

26. [*M32*] (А. Шенхаж (A. Schönhage), 1974.) (a) Докажите в обозначениях упр. 14, что $U_t(n) \ge \min(2 + U_t(n-1), 2 + U_{t-1}(n-1))$ при $n \ge 3$. [*Указание.* Сконструируйте соперника посредством уменьшения количества элементов с n до $n - 1$ до тех пор, пока частичное упорядочение не будет состоять полностью из компонентов вида $\bullet$ или $\bullet\!\!\longrightarrow\!\!\bullet$.] (b) Аналогично докажите, что

$$U_t(n) \ge \min(2 + U_t(n-1), 3 + U_{t-1}(n-1), 3 + U_t(n-2))$$

при $n \ge 5$, сконструировав соперника, который имеет дело с компонентами $\bullet$, $\bullet\!\!\longrightarrow\!\!\bullet$, $\bullet\!\!\longrightarrow\!\!\bullet\!\!\longrightarrow\!\!\bullet$, $\bullet\!\!\longrightarrow\!\!\bullet\!\!\longrightarrow\!\!\bullet\!\!\longrightarrow\!\!\bullet$. (c) Таким образом, получим $U_t(n) \ge n + t + \min(\lfloor (n-t)/2 \rfloor, t) - 3$ при $1 \le t \le n/2$. [Неравенства в (a) и (b) относятся также к случаю, когда V или W заменяет U, обеспечивая таким образом оптимальность определенных элементов в табл. 1.]

▸ **27.** [*M34*] *Рандомизированный соперник* — это соперник, алгоритм поведения которого позволяет при принятии решения использовать подбрасывание монеты.

a) Пусть A — рандомизированный соперник и пусть $\Pr(l)$ — вероятность того, что A достигнет листа l данного дерева сравнений. Покажите, что, если $\Pr(l) \le p$ для всех l, высота дерева сравнений $\ge \lg(1/p)$.

b) Проанализируйте поведение следующего соперника для выбора t-го по старшинству из n элементов, причем целочисленные параметры q и r будут выбраны позже.

A1. Выбирается случайное множество T, состоящее из t элементов; все $\binom{n}{t}$ вариантов считаются равновероятными. (Будем считать, что обеспечивается условие, при котором $t - 1$ наибольших элементов принадлежат T.) Пусть $S = \{1, \ldots, n\} \setminus T$ — другие элементы и установлено $S_0 \leftarrow S$, $T_0 \leftarrow T$; S_0 и T_0 будут представлять элементы, которые могут стать t-ми по старшинству.

A2. До тех пор, пока $|T_0| > r$, все сравнения $x : y$ выполняются следующим образом. Если $x \in S$ и $y \in T$, считается, что $x < y$. Если $x \in S$ и $y \in S$, подбрасывается монета и удаляется меньший элемент из S_0, если он принадлежит S_0. Если $x \in T$ и $y \in T$, подбрасывается монета и удаляется больший элемент из T_0, если он принадлежит T_0.

A3. Как только получим $|T_0| = r$, элементы разделятся на три класса, P, Q, R, следующим образом. Если $|S_0| < q$, считается, что $P = S$, $Q = T_0$, $R = T \setminus T_0$. В противном случае для каждого $y \in T_0$ считается, что $C(y)$ — это элемент из S, который уже прошел сравнение с y и выбирается y_0, такой, что $|C(y_0)|$ минимально. Пусть $P = (S \setminus S_0) \cup C(y_0)$, $Q = (S_0 \setminus C(y_0)) \cup \{y_0\}$, $R = T \setminus \{y_0\}$. Все последующие сравнения $x : y$ выполняются так, что элементы из P считаются меньше элементов из Q, а элементы из Q меньше элементов из R; если же x и y принадлежат одному и тому же классу, подбрасывается монета.

Докажите что, если $1 \le r \le t$ и $|C(y_0)| \le q - r$ в начале шага A3, каждый лист достигается с вероятностью $\le (n + 1 - t)/(2^{n-q}\binom{n}{t})$. *Указание.* Покажите, что будет сделано не менее $n - q$ подбрасываний монеты.

c) Продолжая (b), покажите, что

$$V_t(n) \geq \min(n-1+(r-1)(q+1-r),\ n-q+\lg(\binom{n}{t}/(n+1-t)))$$

для всех целочисленных значений q и r.

d) Выведите (14), выбрав соответственно q и r.

*5.3.4. Сети сортировки

В настоящем разделе мы будем изучать класс методов сортировки, удовлетворяющих некоторому ограничению. Интерес к таким методам объясняется в основном приложениями и солидной теоретической основой. Это новое ограничение требует, чтобы последовательность сравнений *не зависела от предыстории* в том смысле, что если мы сравниваем K_i и K_j, то последующие сравнения для случая $K_i < K_j$ в точности те же, что и для случая $K_i > K_j$, однако i и j меняются ролями.

На рис. 43, (a) изображено дерево сравнений, в котором это условие однородности выполнено. Заметим, что на каждом уровне производится одинаковое число сравнений, поэтому после m сравнений имеется 2^m результатов. Так как $n!$ не является степенью 2, некоторые сравнения будут излишними в том смысле, что одно из их поддеревьев никогда не встречается на практике. Иными словами, на некоторых ветвях дерева приходится выполнять больше сравнений, чем необходимо, чтобы сортировка была правильной на всех соответствующих ветвях.

Поскольку каждый путь на таком дереве сверху донизу определяет все дерево, эту схему сортировки проще изображать в виде *сети*, как на рис. 43, (b). Прямоугольник в подобной сети представляет "модуль компаратора", имеющий два входа (изображены линиями, входящими в модуль сверху) и два выхода (изображены линиями, выходящими вниз); левый выход — меньший из двух входов, а правый выход — больший из них. Элемент K_1' в нижней части сети есть наименьший из $\{K_1, K_2, K_3, K_4\}$, K_2' — второй в порядке возрастания и т. д. Нетрудно доказать, что любая сеть сортировки соответствует дереву сравнений, обладающему свойством независимости от предыстории (в указанном выше смысле), и что любое такое дерево соответствует сети модулей компараторов.

Между прочим, технически модуль компаратора довольно легко изготовить. Предположим, например, что по линиям связи в модуль поступают двоичные числа по одному разряду в единицу времени, начиная со старшего. Каждый модуль компаратора имеет три состояния и функционирует следующим образом.

Момент t		Момент $(t+1)$	
Состояние	Входы	Состояние	Выходы
0	0 0	0	0 0
0	0 1	1	0 1
0	1 0	2	0 1
0	1 1	0	1 1
1	x y	1	x y
2	x y	2	y x

Первоначально все модули находятся в состоянии 0 и выдают 0 0. Модуль переходит в состояние 1 или 2, как только его входы становятся различными. Числа, которые в момент времени t начали поступать сверху в сеть, соответствующую рис. 43, (b),

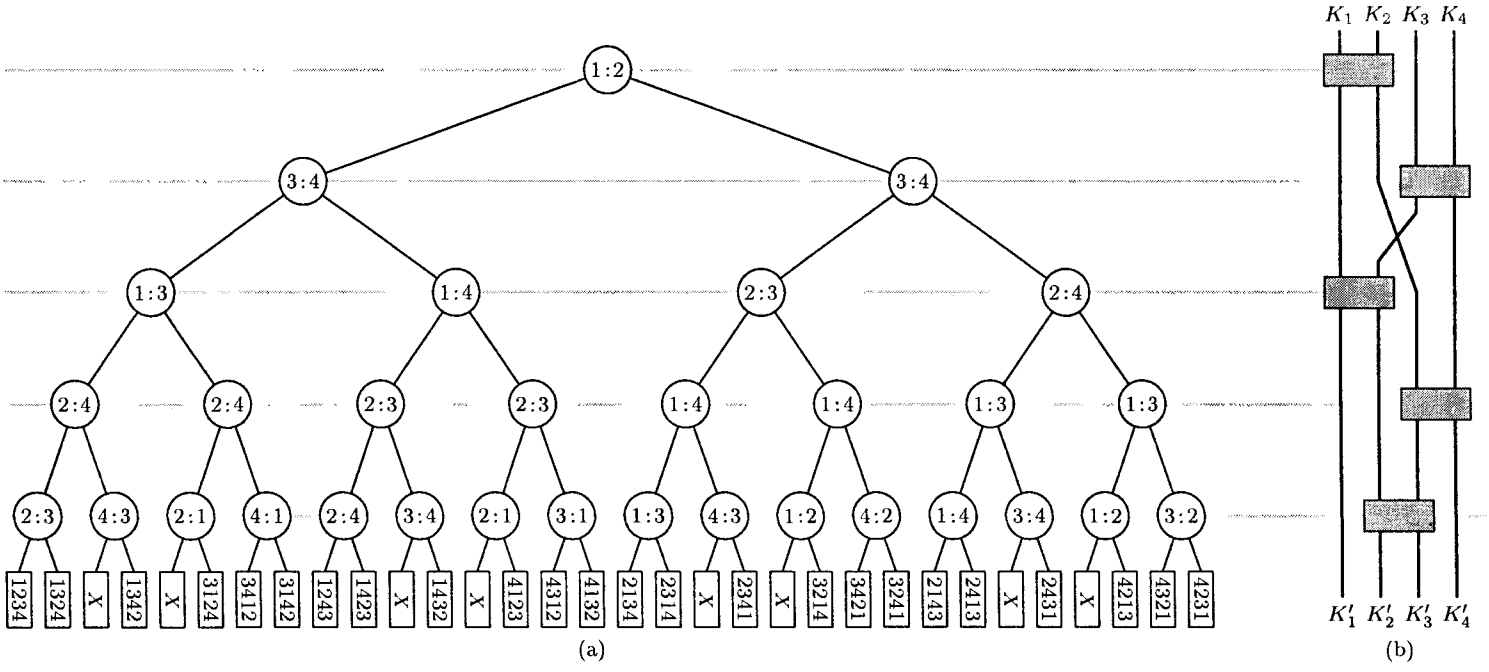


Рис. 43. (a) Дерево сравнений, в котором не учитывается предыстория. (b) Соответствующая ему сеть.

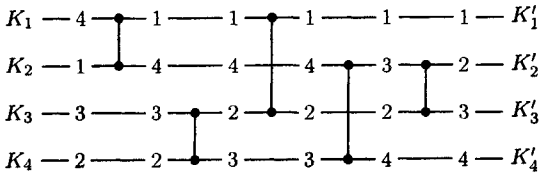


Рис. 44. Еще один способ представления сортировки последовательности $(4, 1, 3, 2)$ посредством сети, изображенной на рис. 43.

начнут в момент $t+3$ выводиться снизу в рассортированном порядке, если включить соответствующий элемент задержки на линиях K'_1 и K'_4.

При разработке теории сетей сортировки удобно изображать их несколько иным способом. На рис. 44 числа поступают *слева*, а модули компараторов представлены в виде вертикальных соединений между двумя прямыми; каждый компаратор вызывает, если необходимо, перестановку своих входов таким образом, что после прохождения компаратора большее число оказывается на *нижней* линии. В правой части диаграммы все числа упорядочены сверху вниз.

Ранее, изучая методы оптимальной сортировки, мы уделяли основное внимание минимизации числа сравнений, почти (или совсем) не учитывая перемещение данных либо сложность структуры принятия решений, которые связаны с таким методом сортировки. В этом отношении сети сортировки имеют некоторое преимущество, так как данные могут храниться в n ячейках, а структура принятия решений "прямолинейна"; нет необходимости запоминать результаты предыдущих сравнений — план неизменен и фиксирован заранее. Еще одним важным преимуществом сетей сортировки является то, что часть операций можно совмещать, т. е. выполнять их параллельно (если в нашем распоряжении имеется компьютер с соответствующей архитектурой). Например, пять шагов на рис. 43 и 44 сокращаются до трех, если организовать одновременные неперекрывающиеся операции сравнения, так как можно объединить первые два и следующие два шага. Ниже в данном разделе мы используем это свойство сетей сортировки. Таким образом, сети сортировки могут оказаться очень полезными на практике, хотя возможность построения эффективной сети сортировки n элементов при больших n вовсе не очевидна; возможно, мы обнаружим, что для поддержания однородной структуры решений требуется много дополнительных сравнений.

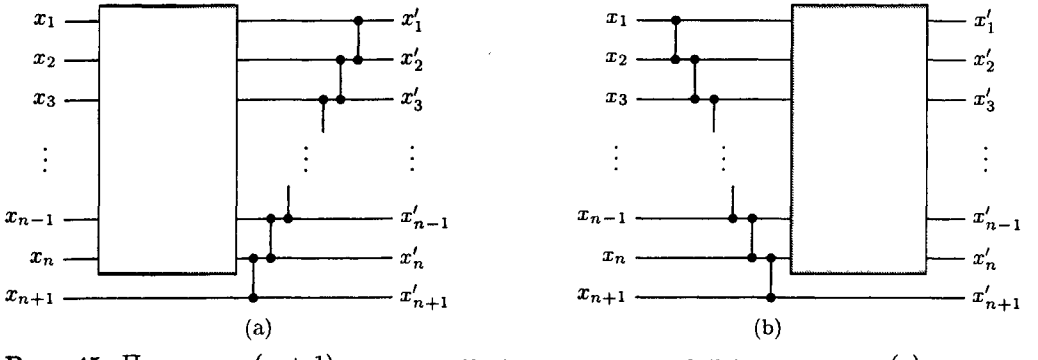


Рис. 45. Получение $(n+1)$-элементных сортировщиков из n-элементных: (a) вставка, (b) выбор.

Если дана сеть для n элементов, имеется два простых способа построения сети сортировки для $n+1$ элементов: один — с использованием принципа *вставки*, а другой — принципа *выбора*. На рис. 45, (a) показано, как $(n+1)$-й элемент может быть вставлен на нужное место после того, как первые n элементов рассортированы, а на рис. 45, (b) показано, как можно выбрать наибольший элемент, прежде чем перейти к сортировке остальных элементов. Многократное применение процедуры, показанной на рис. 45, (a), позволяет получить сетевой аналог метода простых вставок (алгоритм 5.2.1S), а многократное применение процедуры на рис. 45, (b) приводит к сетевому аналогу метода пузырька (алгоритм 5.2.2B).

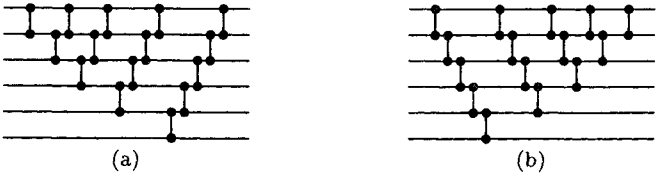


Рис. 46. Сетевые аналоги элементарных схем внутренней сортировки, которые получены в результате многократного применения операции, представленной на рис. 45: (a) простая вставка, (b) метод пузырька.

На рис. 46 изображены соответствующие сети для шести элементов. Интересно заметить, что если сжать каждую сеть, чтобы обеспечить выполнение одновременных операций, то оба метода сведутся к одной и той же "треугольной" процедуре с $(2n-3)$ стадиями (рис. 47).

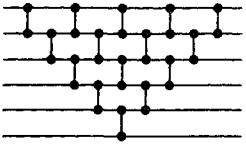
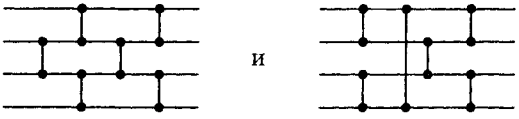


Рис. 47. При параллельном выполнении операций метод простой вставки совпадает с методом пузырька!

Легко доказать, что сети, представленные на рис. 43 и 44, позволяют сортировать любое множество из четырех чисел, поскольку первые четыре компаратора направляют наименьший и наибольший элементы на положенные им места, а последний компаратор располагает в требуемом порядке остальные два элемента. Однако не всегда так легко сказать, будет ли данная сеть сортировать все возможные входные последовательности; например, сети



являются правильными четырехэлементными сетями сортировки, но доказательство их правильности отнюдь не тривиально. Конечно, для этого достаточно проверить каждую n-элементную сеть на всех $n!$ перестановках n различных чисел, но фактически мы можем обойтись значительно меньшим количеством проверок.

Теорема Z (Принцип нулей и единиц). *Если сеть с n входами сортирует в порядке неубывания все 2^n последовательностей из 0 и 1, то она будет сортировать в том же порядке любую последовательность n чисел.*

Доказательство. (Это частный случай теоремы Бурисиуса (Bouricius); см. упр. 5.3.1–12.) Если $f(x)$ — любая монотонная функция, для которой $f(x) \le f(y)$ при $x \le y$, и если данная сеть преобразует $\langle x_1, \ldots, x_n\rangle$ в $\langle y_1, \ldots, y_n\rangle$, то, как нетрудно видеть, эта сеть преобразует $\langle f(x_1), \ldots, f(x_n)\rangle$ в $\langle f(y_1), \ldots, f(y_n)\rangle$. Если $y_i > y_{i+1}$ при некотором i, то рассмотрим монотонную функцию f, которая для всех чисел $< y_i$ принимает значение 0, а для всех чисел $\ge y_i$ — значение 1. Эта функция определяет последовательность нулей и единиц $\langle f(x_1), \ldots, f(x_n)\rangle$, которая не сортируется данной сетью. Значит, если все последовательности 0–1 поддаются сортировке, то будем иметь $y_i \le y_{i+1}$ для всех $1 \le i < n$. ▌

Принцип нулей и единиц довольно полезен для построения сетей сортировки. В качестве нетривиального примера можно с его помощью вывести обобщенный вариант "обменной сортировки со слиянием" Бэтчера (алгоритм 5.2.2M). Идея состоит в том, чтобы сортировать $m+n$ элементов, сортируя первые m и последние n элементов независимо, а затем "пропустить" результат через (m,n)-*сеть слияния*. Построить (m,n)-сеть слияния можно по индукции следующим образом.

a) Если $m = 0$ или $n = 0$, то сеть пуста. Если $m = n = 1$, то сеть состоит из единственного модуля компаратора.

b) Если $mn > 1$, обозначим сливаемые последовательности через $\langle x_1, \ldots, x_m\rangle$ и $\langle y_1, \ldots, y_n\rangle$. Сольем "нечетные последовательности" $\langle x_1, x_3, \ldots, x_{2\lceil m/2\rceil-1}\rangle$ и $\langle y_1, y_3, \ldots, y_{2\lceil n/2\rceil-1}\rangle$ и получим рассортированный результат $\langle v_1, v_2, \ldots, v_{\lceil m/2\rceil+\lceil n/2\rceil}\rangle$; сольем "четные последовательности" $\langle x_2, x_4, \ldots, x_{2\lfloor m/2\rfloor}\rangle$ и $\langle y_2, y_4, \ldots, y_{2\lfloor n/2\rfloor}\rangle$ и получим рассортированный результат $\langle w_1, w_2, \ldots, w_{\lfloor m/2\rfloor+\lfloor n/2\rfloor}\rangle$. И наконец, применим операции сравнения-обмена

$$w_1 : v_2, \quad w_2 : v_3, \quad w_3 : v_4, \quad \ldots, \quad w_{\lfloor m/2\rfloor+\lfloor n/2\rfloor} : v^* \tag{1}$$

к последовательности

$$\langle v_1, w_1, v_2, w_2, v_3, w_3, \ldots, v_{\lfloor m/2\rfloor+\lfloor n/2\rfloor}, w_{\lfloor m/2\rfloor+\lfloor n/2\rfloor}, v^*, v^{**}\rangle. \tag{2}$$

Теперь результат будет рассортирован. (!) Здесь $v^* = v_{\lfloor m/2\rfloor+\lfloor n/2\rfloor+1}$ не существует, если m и n оба четные, и $v^{**} = v_{\lfloor m/2\rfloor+\lfloor n/2\rfloor+2}$ существует, лишь если m и n оба нечетные; общее число модулей компараторов, указанных в (1), равно $\lfloor (m+n-1)/2\rfloor$.

Назовем (m,n)-сеть слияния Бэтчера *четно-нечетным слиянием*. Построенное в соответствии с этими принципами $(4,7)$-слияние показано на рис. 48.

Чтобы доказать, что эта довольно странная процедура действительно работает при $mn > 1$, воспользуемся принципом нулей и единиц и проверим ее на всех последовательностях 0 и 1. После начальных сортировок m и n последовательность $\langle x_1, \ldots, x_m\rangle$ будет состоять из k нулей, за которыми следуют $m-k$ единиц, а последовательность $\langle y_1, \ldots, y_m\rangle$ — из l нулей с последующими $n-l$ единицами при некоторых k и l. Значит, последовательность $\langle v_1, v_2, \ldots\rangle$ будет состоять из точно $\lceil k/2\rceil + \lceil l/2\rceil$ нулей с последующими единицами, а $\langle w_1, w_2 \ldots\rangle$ — из $\lfloor k/2\rfloor + \lfloor l/2\rfloor$

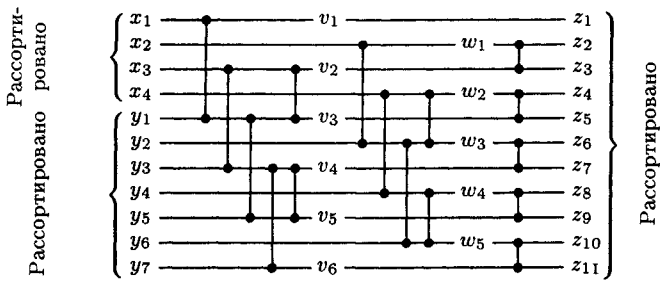


Рис. 48. Четно-нечетное слияние для $m = 4$ и $n = 7$.

нулей с последующими единицами. Решающим моментом доказательства является то, что

$$\left(\lceil k/2\rceil + \lceil l/2\rceil\right) - \left(\lfloor k/2\rfloor + \lfloor l/2\rfloor\right) = 0,\ 1 \text{ или } 2. \tag{3}$$

Если эта разность равна 0 или 1, то последовательность (2) уже упорядочена, а если она равна 2, то одна из операций сравнения-обмена в (1) ставит все на свои места. Доказательство завершено. (Заметим, что принцип нулей и единиц сводит $\binom{m+n}{m}$ случаев в задаче слияния всего лишь к $(m+1)(n+1)$ случаям, каждый из которых представляется двумя параметрами — k и l.)

Пусть $C(m,n)$ — число модулей компараторов, используемых при четно-нечетном слиянии m- и n-элементов, не считая начальных m- и n-сортировок; имеем

$$C(m,n) = \begin{cases} mn, & \text{если } mn \le 1; \\ C\left(\lceil m/2\rceil, \lceil n/2\rceil\right) + C\left(\lfloor m/2\rfloor, \lfloor n/2\rfloor\right) + \lfloor (m+n-1)/2\rfloor, & \text{если } mn > 1. \end{cases} \tag{4}$$

В общем случае это не слишком простая функция от m и n, однако, заметив, что $C(1,n) = n$ и что

$$\begin{aligned} &C(m+1,n+1) - C(m,n) \\ &\qquad = 1 + C\left(\lfloor m/2\rfloor + 1, \lfloor n/2\rfloor + 1\right) - C\left(\lfloor m/2\rfloor, \lfloor n/2\rfloor\right), \quad \text{если } mn \ge 1, \end{aligned}$$

мы можем вывести соотношение

$$C(m+1,n+1) - C(m,n) = \lfloor \lg m\rfloor + 2 + \lfloor n/2^{\lfloor \lg m\rfloor + 1}\rfloor, \quad \text{если } n \ge m \ge 1. \tag{5}$$

Следовательно,

$$C(m, m+r) = B(m) + m + R_m(r) \qquad \text{при } m \ge 0 \text{ и } r \ge 0, \tag{6}$$

где $B(m) = \sum_{k=1}^{m} \lceil \lg k\rceil$ — это функция "бинарной вставки" из соотношения 5.3.1–(3), а $R_m(r)$ обозначает сумму первых m членов ряда

$$\left\lfloor \frac{r+0}{1}\right\rfloor + \left\lfloor \frac{r+1}{2}\right\rfloor + \left\lfloor \frac{r+2}{4}\right\rfloor + \left\lfloor \frac{r+3}{4}\right\rfloor + \left\lfloor \frac{r+4}{8}\right\rfloor + \cdots + \left\lfloor \frac{r+j}{2^{\lfloor \lg j\rfloor + 1}}\right\rfloor + \cdots. \tag{7}$$

Если же $r = 0$, получаем важный частный случай

$$C(m,m) = B(m) + m. \tag{8}$$

Кроме того, если $t = \lceil \lg m\rceil$, то

$$\begin{aligned} R_m(r + 2^t) &= R_m(r) + 1\cdot 2^{t-1} + 2\cdot 2^{t-2} + \cdots + 2^{t-1}\cdot 2^0 + m \\ &= R_m(r) + m + t\cdot 2^{t-1}. \end{aligned}$$

Следовательно, $C(m, n+2^t) - C(m,n)$ имеет простой вид и

$$C(m,n) = \left(\frac{t}{2} + \frac{m}{2^t}\right) n + O(1) \qquad \text{при фиксированном } m,\ n \to \infty,\ t = \lceil \lg m \rceil; \quad (9)$$

член $O(1)$ становится, в конце концов, периодической функцией от n с периодом 2^t. Асимптотически при $n \to \infty$ величина $C(n,n) = n \lg n + O(n)$, как следует из (8) и упр. 5.3.1–15.

Сети с минимальным числом сравнений. Пусть $\hat{S}(n)$ — минимальное число сравнений, требуемых в сети сортировки для n элементов; ясно, что $\hat{S}(n) \geq S(n)$, где $S(n)$ — минимальное число сравнений, необходимое для сортировки безо всяких ограничений (см. раздел 5.3.1). Мы видели, что $\hat{S}(4) = 5 = S(4)$, поэтому новое ограничение не приведет к потере эффективности при $n = 4$; но уже при $n = 5$ оказывается, что $\hat{S}(5) = 9$, в то время как $S(5) = 7$. Задача определения $\hat{S}(n)$ кажется еще более трудной, чем задача определения $S(n)$; до сих пор неизвестно даже асимптотическое поведение $\hat{S}(n)$.

Интересно проследить историю поиска путей решения этой задачи, так как каждый новый шаг стоил определенных усилий. Сети сортировки были впервые исследованы Ф. Н. Армстронгом (P. N. Armstrong), Р. Дж. Нельсоном (R. J. Nelson) и Д. Дж. О'Коннором (D. J. O'Connor) около 1954 года [см. *U. S. Patent 3029413*]. Как сказано в их патентной заявке, "приложив старания, можно сконструировать экономичные n-входные сортирующие переключатели с помощью уменьшенного числа двухвходовых сортирующих переключателей". Показав, что $\hat{S}(n+1) \leq \hat{S}(n) + n$, они предложили специальные конструкции для $4 \leq n \leq 8$, использовав соответственно 5, 9, 12, 18 и 19 компараторов. Работая далее над этой задачей, Нельсон совместно с Р. Ч. Бозе (R. C. Bose) задались целью показать, что $\hat{S}(2^n) \leq 3^n - 2^n$ при всех n; следовательно, $\hat{S}(n) = O(n^{\lg 3}) = O(n^{1.585})$. Бозе и Нельсон опубликовали свой интересный метод в *JACM* **9** (1962), 282–296, где высказали предположение, что это наилучший возможный результат; Т. Н. Хиббард (T. N. Hibbard) в *JACM* **10** (1963), 142–150, описал аналогичный, но более простой метод, в котором используется такое же число сравнений, подкрепив тем самым это предположение.

В 1964 году Р. У. Флойд (R. W. Floyd) и Д. Э. Кнут использовали новый подход к этой задаче, приведший к асимптотической оценке вида $\hat{S}(n) = O\big(n^{1+c/\sqrt{\log n}}\big)$. Независимо К. Э. Бэтчер (K. E. Batcher) разработал описанную выше общую стратегию слияния. Используя компараторы, число которых определяется рекурсивным выражением

$$c(1) = 0, \quad c(n) = c\big(\lceil n/2 \rceil\big) + c\big(\lfloor n/2 \rfloor\big) + C\big(\lceil n/2 \rceil, \lfloor n/2 \rfloor\big) \quad \text{при } n \geq 2, \qquad (10)$$

он доказал, что (см. упр. 5.2.2–14)

$$c(2^t) = (t^2 - t + 4)2^{t-2} - 1;$$

следовательно, $\hat{S}(n) = O\big(n(\log n)^2\big)$. Как Бэтчер, так и Флойд с Кнутом опубликовали свои конструкции лишь через некоторое время [см. *Notices of the Amer. Math. Soc.* **14** (1967), 283; *Proc. AFIPS Spring Joint Computer Conf.* **32** (1968), 307–314].

Кое-кому удалось сократить число компараторов, используемых в конструкции слияния с обменами, предложенной Бэтчером. В (11) показаны наилучшие из известных в настоящее время верхних оценок для $\hat{S}(n)$.

$$
\begin{array}{rcccccccccccccccc}
n & = & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 \\
\mathrm{c}(n) & = & 0 & 1 & 3 & 5 & 9 & 12 & 16 & 19 & 26 & 31 & 37 & 41 & 48 & 53 & 59 & 63 \\
\hat{S}(n) & \le & 0 & 1 & 3 & 5 & 9 & 12 & 16 & 19 & 25 & 29 & 35 & 39 & 45 & 51 & 56 & 60
\end{array}
\tag{11}
$$

Так как $\hat{S}(n) < \mathrm{c}(n)$ при $8 < n \le 16$, обменная сортировка со слиянием неоптимальна при всех $n > 8$. Если $n \le 8$, то такая сортировка эквивалентна по количеству компараторов конструкции Бозе и Нельсона. Флойд и Кнут доказали в 1964–1966 годах, что указанные значения $\hat{S}(n)$ *точны* при $n \le 8$ [см. *A Survey of Combinatorial Theory* (North-Holland, 1973), 163–172]; значения $\hat{S}(n)$ при $n > 8$ до сих пор неизвестны.

Конструкции, позволившие получить указанные выше значения (см. (11)), представлены графически на рис. 49. Сеть при $n = 9$, основанная на интересном трехпутевом слиянии, была построена Р. У. Флойдом в 1964 году; установить ее корректность можно при помощи общего принципа, описанного в упр. 27. Сеть при $n = 10$ построил в 1969 году А. Ваксман (A. Waksman); он рассматривал входы как перестановки множества $\{1, 2, \ldots, 10\}$ и пытался, сохраняя некоторую симметрию, насколько возможно уменьшить число значений, которые могут появляться в каждой строке на данной стадии.

Представленный вариант сети для $n = 13$ построен по довольно отличающейся технологии: Хью Жуиле (Hugues Juillé) разработал программу, чтобы построить эту сеть, моделируя эволюционный процесс размножения [*Lecture Notes in Comp. Sci.* **929** (1995), 246–260]. Сеть имеет несколько необычный вид, но она работает; к тому же она меньше других сетей, созданных рациональным человеческим разумом.

В 1969 году Дж. Шапиро (G. Shapiro) нашел сеть сортировки 16 элементов с 62 компараторами, и это было весьма неожиданно, поскольку результат, полученный методом Бэтчера (63 сравнения), казалось, невозможно улучшить, если n является степенью 2. Познакомившись с конструкцией Шапиро, в еще большее изумление поверг всех М. У. Грин (M. W. Green), который нашел сортировку с 60 сравнениями, показанную на рис. 49. Первая часть конструкции Грина довольно проста для понимания; после того как выполнены 32 операции сравнения-обмена слева от пунктирной линии, все прямые можно так пометить 16 подмножествами $\{a, b, c, d\}$, чтобы о прямой, помеченной s, было известно, что она содержит числа, меньшие или равные содержимому прямой, помеченной t, всякий раз, когда s есть подмножество t. Состояние сортировки на этой стадии более подробно рассматривается в упр. 32. Однако сравнения, выполняемые на последующих уровнях сети Грина, становятся совершенно загадочными. До сих пор никто не знает, как обобщить эту конструкцию, чтобы получить столь же эффективные сети для больших значений n.

Шапиро и Грин создали также изображенную на рис. 49 сеть для $n = 12$. Хорошие сети для $n = 11$, 14 или 15 можно получить, удалив нижнюю линию в сети для $n + 1$ вместе со всеми компараторами, подсоединенными к этой линии.

Наилучшая известная к настоящему моменту сеть для 256 элементов, разработанная Д. Ван Ворисом (D. Van Voorhis), свидетельствует, что $\hat{S}(256) \le 3651$ по сравнению с 3839 в сети, которая построена по методу Бэтчера. [См. R. L. Drysdale, F. H. Young, *SICOMP* **4** (1975), 264–270.] При $n \to \infty$ оказывается, что $\hat{S}(n) = O(n \log n)$; эта впечатляющая верхняя оценка найдена Айтаи (Ajtai), Комлошем (Komlós) и Шемереди (Szemerédi) в *Combinatorica* **3** (1983), 1–19. Построенные ими сети не представляют практического интереса, поскольку множество компараторов

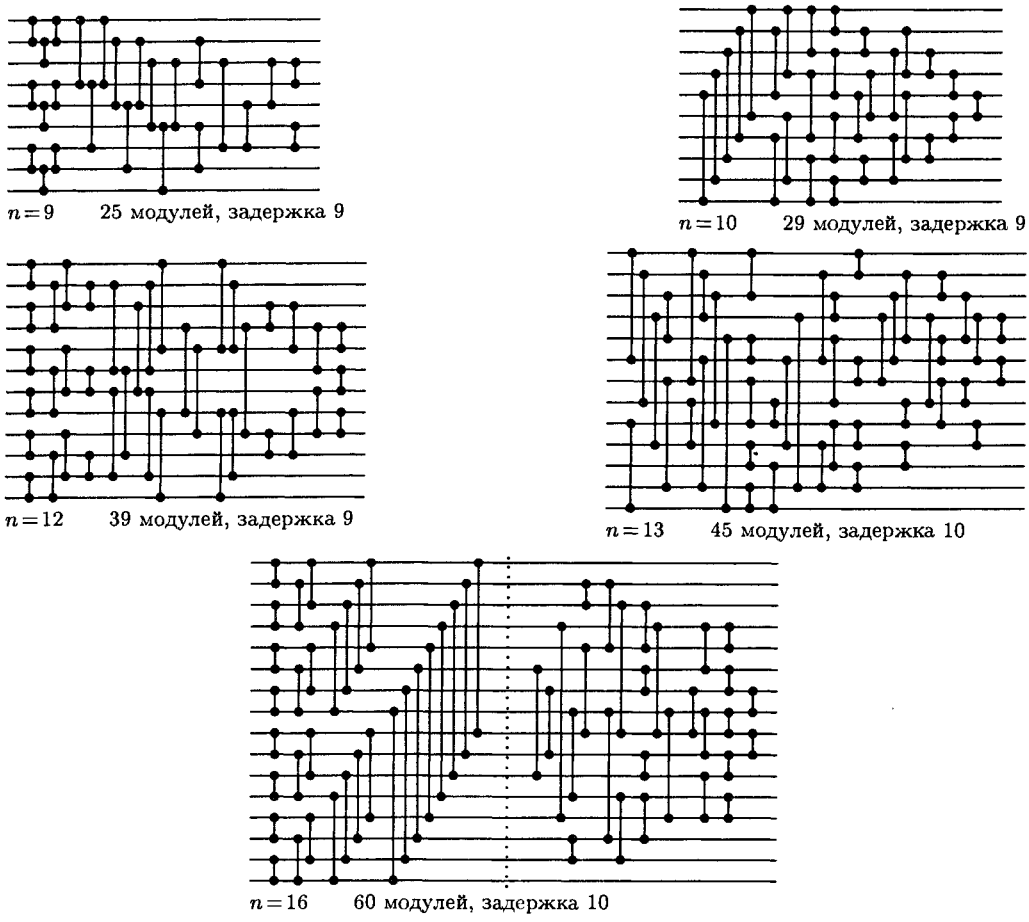


Рис. 49. Эффективные сети сортировки.

используется в них только для того, чтобы сохранить множитель $\log n$; метод Бэтчера работает гораздо лучше до тех пор, пока n не выйдет за пределы доступной памяти на всех компьютерах старушки Земли! Но теорема Айтаи, Комлоса и Шемереди все-таки правильно устанавливает асимптотический рост $\hat{S}(n)$ до уровня постоянного множителя.

Сети с минимальным временем. В физических реализациях сетей сортировки и на компьютерах с параллельной архитектурой можно выполнять непересекающиеся операции сравнения-обмена одновременно, поэтому кажется естественным попытаться минимизировать время задержки. После некоторого размышления приходим к выводу, что время задержки сети сортировки равно максимальному числу компараторов, расположенных на каком-либо "пути" через сеть, если определить путь как траекторию любого движения слева направо, возможно, с переходом с одной линии на другую через компараторы. У каждого компаратора мы можем поставить порядковый номер, указывающий самый ранний момент, когда может быть выполнено сравнение; этот номер на единицу больше, чем максимальный номер

у компараторов, предшествующих данному (рис. 50, (a); в части (b) этого рисунка показана та же сеть, перерисованная так, чтобы каждое сравнение выполнялось как можно раньше.)

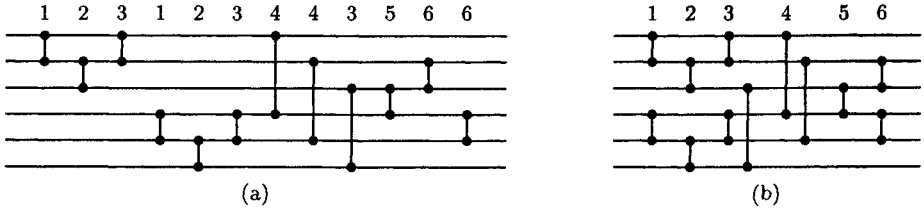


Рис. 50. Выполнение каждого сравнения как можно раньше по времени.

В описанной выше сети Бэтчера для четно-нечетного слияния затрачивается $T_B(m,n)$ единиц времени, где $T_B(m,0) = T_B(0,n) = 0$, $T_B(1,1) = 1$, и

$$T_B(m,n) = 1 + \max\bigl(T_B(\lfloor m/2\rfloor, \lfloor n/2\rfloor), T_B(\lceil m/2\rceil, \lceil n/2\rceil)\bigr) \quad \text{при } mn \geq 2.$$

Используя эти соотношения, можно доказать по индукции, что

$$T_B(m,n+1) \geq T_B(m,n);$$

следовательно, $T_B(m,n) = 1 + T_B\bigl(\lceil m/2\rceil, \lceil n/2\rceil\bigr)$ для $mn \geq 2$; отсюда заключаем, что

$$T_B(m,n) = 1 + \lceil \lg \max(m,n) \rceil \qquad \text{при } mn \geq 1. \tag{12}$$

Таким образом, как показано в упр. 5, метод сортировки Бэтчера имеет время задержки

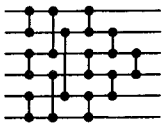
$$\binom{1 + \lceil \lg n \rceil}{2}. \tag{13}$$

Пусть $\hat{T}(n)$ — минимальное время задержки, достижимое в любой сети сортировки n элементов. Некоторые из описанных выше сетей можно так улучшить, не используя дополнительных компараторов, чтобы они имели меньшее время задержки, как показано на рис. 51 для $n = 6$ и $n = 9$, а в упр. 7 — для $n = 10$. Можно получить еще меньшее время задержки, если добавить один или два дополнительных модуля, как показано на рис. 51 (см. сети для $n = 10$, 12 и 16). Эти схемы приводят к следующим верхним оценкам для $\hat{T}(n)$ при малых значениях n:

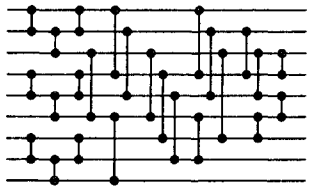
$$\begin{array}{rcccccccccccccccc} n & = & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 \\ \hat{T}(n) & \leq & 0 & 1 & 3 & 3 & 5 & 5 & 6 & 6 & 7 & 7 & 8 & 8 & 9 & 9 & 9 & 9 \end{array} \tag{14}$$

Известно, что приведенные здесь значения точны при $n \leq 10$ (см. упр. 4). Сети, изображенные на рис. 51, заслуживают тщательного изучения, поскольку вовсе не очевидно, что они годятся для сортировки; эти сети были созданы в 1969–1971 годах Дж. Шапиро ($n = 6$, 9, 12) и Д. Ван Ворисом ($n = 10$, 16).

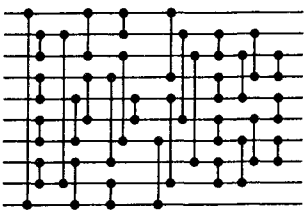
Сети слияния. Пусть $\hat{M}(m,n)$ обозначает минимальное число компараторов, необходимых для сети, которая сливает m элементов $x_1 \leq \cdots \leq x_m$ с n элементами $y_1 \leq \cdots \leq y_n$, образуя рассортированную последовательность $z_1 \leq \cdots \leq z_{m+n}$. К настоящему времени не создано ни одной сети слияния, которая была бы лучше



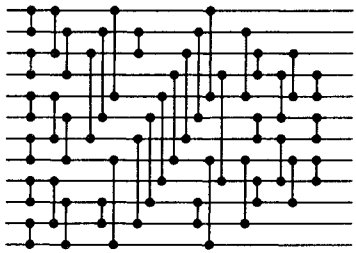
$n=6$ 12 модулей, задержка 5



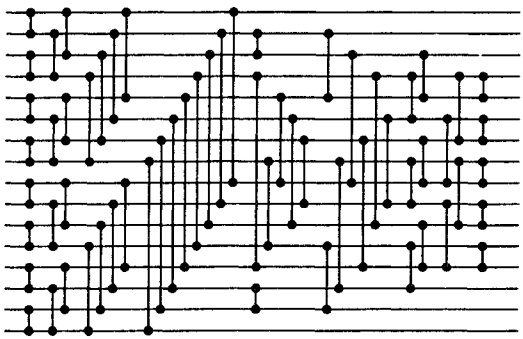
$n=9$ 25 модулей, задержка 8



$n=10$ 31 модуль, задержка 7



$n=12$ 40 модулей, задержка 8



$n=16$ 61 модуль, задержка 9

Рис. 51. Наиболее быстродействующие сети сортировки, предназначенные для параллельных вычислений.

описанной выше сети четно-нечетного слияния; следовательно, функция $C(m,n)$ в (6) представляет наилучшую известную верхнюю оценку для $\hat{M}(m,n)$.

Р. У. Флойд обнаружил интересный способ, позволяющий определить *нижние* оценки в этой задаче слияния.

Теорема F. $\hat{M}(2n,2n) \geq 2\hat{M}(n,n) + n$ *для всех* $n \geq 1$.

Доказательство. Рассмотрим сеть с $\hat{M}(2n,2n)$ модулями компараторов, способную сортировать все входные последовательности $\langle z_1, \ldots, z_{4n}\rangle$, такие, что $z_1 \leq z_3 \leq \cdots \leq z_{4n-1}$ и $z_2 \leq z_4 \leq \cdots \leq z_{4n}$. Мы можем считать, что каждый модуль заменяет (z_i, z_j) элементом $\bigl(\min(z_i,z_j), \max(z_i,z_j)\bigr)$ при некоторых $i < j$ (упр. 16). Итак, компараторы можно разделить на три класса:

a) $i \leq 2n$ и $j \leq 2n$;

b) $i > 2n$ и $j > 2n$;

c) $i \leq 2n$ и $j > 2n$.

Класс (a) должен содержать, по крайней мере, $\hat{M}(n,n)$ компараторов, так как z_{2n+1}, $z_{2n+2}, \ldots, z_{4n}$ могут уже находиться на своих местах, когда слияние начинается; аналогично в классе (b) должно быть хотя бы $\hat{M}(n,n)$ компараторов. Кроме того, как показывает входная последовательность $\langle 0,1,0,1,\ldots,0,1\rangle$, класс (c) содержит не менее n компараторов, так как n нулей должны переместиться из $\{z_{2n+1},\ldots,z_{4n}\rangle$ в $\{z_1,\ldots,z_{2n}\rangle$. ▌

Многократное применение теоремы F доказывает, что $\hat{M}(2^m,2^m) \geq \frac{1}{2}(m+2)2^m$; следовательно, $\hat{M}(n,n) \geq \frac{1}{2}n\lg n + O(n)$. Из теоремы 5.3.2M известно, что слияние *без* сетевого ограничения требует лишь $M(n,n) = 2n-1$ сравнений; таким образом, мы доказали, что сетевое слияние сложнее по существу, чем слияние вообще.

Четно-нечетное слияние показывает, что

$$\hat{M}(m,n) \leq C(m,n) = \tfrac{1}{2}(m+n)\lg\min(m,n) + O(m+n).$$

В работе P. B. Miltersen, M. Paterson, J. Tarui, *JACM* **43** (1996), 147–165, теорема F доработана и установлена нижняя оценка

$$\hat{M}(m,n) \geq \tfrac{1}{2}\big((m+n)\lg(m+1) - m/\ln 2\big) \qquad \text{при } 1 \leq m \leq n.$$

Следовательно, $\hat{M}(m,n) = \frac{1}{2}(m+n)\lg\min(m,n) + O(m+n)$.

Точная формула $\hat{M}(2,n) = C(2,n) = \lceil \frac{3}{2}n \rceil$ доказана в работе A. C. Yao, F. F. Yao, *JACM* **23** (1976), 566–571. Также известно, что значение $\hat{M}(m,n)$ равно $C(m,n)$ для $m = n \leq 5$ [см. упр. 9].

Битонная сортировка. Если допустимы одновременные сравнения, то, как видно из формулы (12), при четно-нечетном слиянии для $1 \leq m \leq n$ возникает задержка на $\lceil \lg(2n) \rceil$ единиц времени. Бэтчер изобрел другой тип сети слияния, названный *битонным сортировщиком* (bitonic sorter)*, для которого время задержки снижается до $\lceil \lg(m+n) \rceil$. Но он требует больше модулей компараторов. [См. *U.S. Patent 3428946* (1969).]

Последовательность $\langle z_1,\ldots,z_p\rangle$ из p чисел будем называть *битонной*, если $z_1 \geq \cdots \geq z_k \leq \cdots \leq z_p$ для некоторого k, $1 \leq k \leq p$. (Сравните это с обычным определением "монотонных" последовательностей.) Битонный сортировщик порядка p — это сеть компараторов, способная сортировать в порядке неубывания любую битонную последовательность длиной p. Задача слияния $x_1 \leq \cdots \leq x_m$ с $y_1 \leq \cdots \leq y_n$ является частным случаем задачи битонной сортировки, так как слияние можно осуществить, применив к последовательности $\langle x_m,\ldots,x_1,y_1,\ldots,y_n\rangle$ битонный сортировщик порядка $m+n$.

Заметим, что если последовательность $\langle z_1,\ldots,z_p\rangle$ битонная, то таковыми же являются и все ее подпоследовательности. Вскоре после того, как Бэтчер открыл сети четно-нечетного слияния, он обнаружил, что аналогичным образом можно построить битонный сортировщик порядка p, сначала независимо сортируя битонные подпоследовательности $\langle z_1,z_3,z_5,\ldots\rangle$ и $\langle z_2,z_4,z_6,\ldots\rangle$, а затем выполняя сравнения-обмены $z_1 : z_2$, $z_3 : z_4$, ... (Доказательство приводится в упр. 10.) Если соответству-

---

* Термин "битонная последовательность" (bitonic sequence) введен по аналогии с термином "монотонная последовательность" (monotonic sequence). — *Прим. перев.*

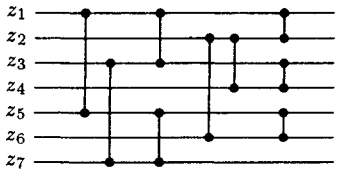


Рис. 52. Битонный сортировщик Бэтчера порядка 7.

ющее число модулей компараторов обозначить через $C'(p)$, то будем иметь

$$C'(p) = C'(\lceil p/2 \rceil) + C'(\lfloor p/2 \rfloor) + \lfloor p/2 \rfloor \qquad \text{при } p \geq 2; \tag{15}$$

а время задержки, очевидно, равно $\lceil \lg p \rceil$. На рис. 52 показан битонный сортировщик порядка 7, построенный этим способом; он может быть использован и как (3, 4)-, и как (2, 5)-сеть слияния с задержкой в три единицы; четно-нечетное слияние для $m = 2$ и $n = 5$ имеет на один компаратор меньше, но на один уровень задержки больше.

Особый интерес представляет битонный сортировщик Бэтчера порядка 2^t; он состоит из t уровней по 2^{t-1} компараторов в каждом. Пронумеруем входные линии $z_0, z_1, \ldots, z_{2^t-1}$; при этом элемент z_i сравнивается с z_j на уровне l тогда и только тогда, когда двоичные представления i и j различаются только в l-м слева разряде. Эта простая структура приводит к созданию параллельной сети сортировки, которая функционирует так же быстро, как обменная сортировка со слиянием (алгоритм 5.2.2M), но реализовать ее значительно проще (см. упр. 11 и 13).

Битонная сортировка оптимальна в том смысле, что ни один метод параллельного слияния, основанный на одновременно несвязанных сравнениях, не может выполнять сортировку быстрее, чем за $\lceil \lg(m+n) \rceil$ стадий, независимо от того, используется ли при этом предыстория (см. упр. 46). Существует и другой способ достижения оптимума, который требует меньше сравнений, но логика его чуть сложнее. Этот метод анализируется в упр. 57.

Если $1 \leq m \leq n$, то n-й (считая от наименьшего) выход (m,n)-сети слияния должен зависеть от $2m + [m < n]$ входов (см. упр. 29). Если его можно вычислить, используя компараторы с l уровнями задержки, то в результат будет вовлечено не более 2^l входов; следовательно, $2^l \geq 2m + [m < n]$ и $l \geq \lceil \lg(2m + [m < n]) \rceil$. Бэтчер показал [Report GER-14122 (Akron, Ohio: Goodyear Aerospace Corporation, 1968)], что это минимальное время задержки достигается, если в сети допускается разветвление, т. е. такое разбиение линий, что одно и то же число в одно и то же время используется несколькими модулями. В качестве примера на рис. 53 изображена сеть ($n = 6$), которая сливает один элемент с n другими после всего двух уровней задержки. Конечно, сети с разветвлением не соответствуют нашим соглашениям; довольно легко понять, что любая $(1,n)$-сеть слияния без разветвления должна иметь время задержки $\lg(n+1)$ или более (см. упр. 45).

Сети выбора. Сети можно применить также к задаче раздела 5.3.3. Пусть $\hat{U}_t(n)$ обозначает минимальное число компараторов, необходимых в сети, которая перемещает t наибольших из n различных входов на t определенных выходных линий; они могут располагаться на этих выходных линиях в произвольном порядке. Пусть $\hat{V}_t(n)$ обозначает минимальное количество компараторов, необходимых для перемещения

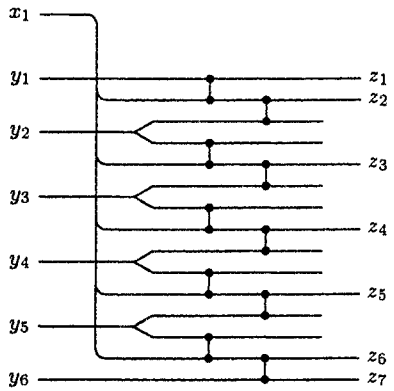


Рис. 53. Сеть, обеспечивающая слияние одного элемента с шестью другими и имеющая разветвления, в результате чего достигается минимальная задержка.

t-го входа в порядке убывания из n различных входов на определенную выходную линию, и пусть $\hat{W}_t(n)$ обозначает минимальное число компараторов, требуемых для перемещения t наибольших из n различных входов на определенные t выходные линии в порядке неубывания. Нетрудно видеть (см. упр. 17), что

$$\hat{U}_t(n) \le \hat{V}_t(n) \le \hat{W}_t(n). \tag{16}$$

Сначала предположим, что имеется $2t$ элементов $\langle x_1, \ldots, x_{2t}\rangle$ и мы хотим выбрать t наибольших. В. Е. Алексеев [*Кибернетика* **5**, 5 (1969), 99–103] заметил, что это может быть выполнено, если сначала рассортировать $\langle x_1, \ldots, x_t\rangle$ и $\langle x_{t+1}, \ldots, x_{2t}\rangle$, а затем сравнить и поменять местами

$$x_1 : x_{2t}, \qquad x_2 : x_{2t-1}, \qquad \ldots, \qquad x_t : x_{t+1}. \tag{17}$$

Так как ни в одной из этих пар не может содержаться более одного из наибольших t элементов (почему?), процедура Алексеева должна выбрать t наибольших элементов.

Если нужно выбрать t наибольших из nt элементов, то мы можем применить эту процедуру $n-1$ раз (исключая каждый раз t элементов); следовательно,

$$\hat{U}_t(nt) \le (n-1)\bigl(2\hat{S}(t) + t\bigr). \tag{18}$$

Алексеев также получил интересную *нижнюю* оценку для задачи выбора.

Теорема А. $\hat{U}_t(n) \ge (n-t)\bigl\lceil \lg(t+1)\bigr\rceil$.

Доказательство. Удобнее рассматривать эквивалентную задачу выбора *наименьших* t элементов. Около каждой линии сети компараторов можно выписать числа (l, u), как показано на рис. 54, где l и u обозначают соответственно минимальное и максимальное значения, которые могут появиться в этом месте, если входом служит перестановка $\{1, 2, \ldots, n\}$. Пусть l_i и l_j — нижние оценки на прямых i и j перед сравнением $x_i : x_j$ и пусть l'_i и l'_j — соответствующие нижние оценки после этого сравнения. Очевидно, что $l'_i = \min(l_i, l_j)$, а в упр. 24 доказывается соотношение (неочевидное)

$$l'_j \le l_i + l_j. \tag{19}$$

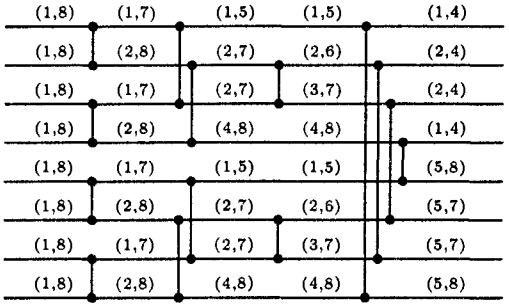


Рис. 54. Отделение четырех наибольших от четырёх наименьших. (Числа над линиями используются в доказательстве теоремы А.)

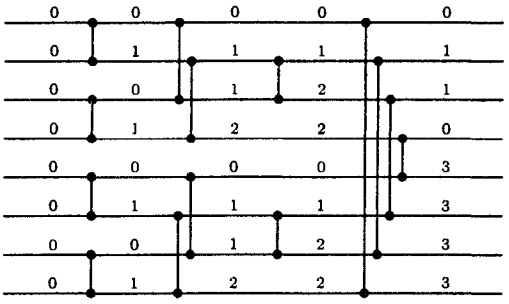


Рис. 55. Иная интерпретация сети, изображенной на рис. 54.

Теперь дадим другую интерпретацию функционирования сети (рис. 55): предположим, что на всех входных прямых содержится нуль и каждый "компаратор" помещает теперь на верхнюю линию меньший из своих входов, а на нижнюю линию — больший вход *плюс единицу*. Получающиеся числа $\langle m_1, m_2, \ldots, m_n \rangle$ обладают свойством

$$2^{m_i} \geq l_i \tag{20}$$

в любом месте сети, так как это свойство первоначально справедливо и сохраняется каждым компаратором в силу (19). Кроме того, окончательное значение $m_1 + m_2 + \cdots + m_n$ равно общему числу компараторов в сети, так как каждый компаратор добавляет к этой сумме единицу.

Если сеть выбирает наименьшие t чисел, то $n - t$ из чисел l_i больше или равны $t + 1$; следовательно, $n - t$ из чисел m_i должны быть $\geq \lceil \lg(t+1) \rceil$. ▌

Нижняя оценка в теореме А оказывается точной, если $t = 1$ или $t = 2$ (см. упр. 19). В табл. 1 даны значения $\hat{U}_t(n)$, $\hat{V}_t(n)$ и $\hat{W}_t(n)$ для небольших t и n. Эндрю Яо (Andrew Yao) [Ph. D. thesis, U. of Illinois (1975)] исследовал асимптотическое поведение $\hat{U}_t(n)$ при фиксированном t и показал, что $\hat{U}_3(n) = 2n + \lg n + O(1)$ и $\hat{U}_t(n) = n\lceil \lg(t+1) \rceil + O\big((\log n)^{\lfloor \lg t \rfloor}\big)$ при $n \to \infty$; минимальное время задержки равно $\lg n + \lfloor \lg t \rfloor \lg\lg n + O(\log\log\log n)$. В работе N. Pippenger, *SICOMP* **20** (1991), 878–887, доказано при помощи неконструктивного метода, что для любого $\epsilon > 0$ существует сеть выбора с $\hat{U}_{\lceil n/2 \rceil}(n) \leq (2+\epsilon) n \lg n$, если только n достаточно велико (в зависимости от ϵ).

Таблица 1

СРАВНЕНИЯ, НЕОБХОДИМЫЕ ДЛЯ СЕТЕЙ ВЫБОРА ($\hat{U}_t(n)$, $\hat{V}_t(n)$, $\hat{W}_t(n)$)

	$t=1$	$t=2$	$t=3$	$t=4$	$t=5$	$t=6$
$n=1$	$(0,0,0)$					
$n=2$	$(1,1,1)$	$(0,1,1)$				
$n=3$	$(2,2,2)$	$(2,3,3)$	$(0,2,3)$			
$n=4$	$(3,3,3)$	$(4,5,5)$	$(3,5,5)$	$(0,3,5)$		
$n=5$	$(4,4,4)$	$(6,7,7)$	$(6,7,8)$	$(4,7,9)$	$(0,4,9)$	
$n=6$	$(5,5,5)$	$(8,9,9)$	$(8,10,10)$	$(8,10,12)$	$(5,9,12)$	$(0,5,12)$

УПРАЖНЕНИЯ (часть 1)

Далее в нескольких упражнениях теория сетей сортировки рассматривается более детально, поэтому будет удобно ввести некоторые обозначения. Вместо модуля сравнения-обмена будем писать $[i:j]$. Сеть с n входами и r модулями компараторов обозначим через $[i_1:j_1]$ $[i_2:j_2]\ldots[i_r:j_r]$, где каждое из i и j меньше или равно n; для краткости будем называть ее *n-сетью*. Сеть называется *стандартной*, если $i_q < j_q$ для $1 \le q \le r$. Так, например, на рис. 44 приведена стандартная 4-сеть, обозначенная последовательностью компараторов $[1:2][3:4][1:3][2:4][2:3]$.

Наши соглашения в тексте о графическом представлении диаграмм сетей позволяют рисовать только стандартные сети; все компараторы $[i:j]$ изображаются прямой линией от i к j, где $i<j$. Чтобы нарисовать нестандартную сеть, можно использовать *стрелку* от i к j, указывающую, что большее число направляется к острию стрелки. Например, на рис. 56 изображена нестандартная сеть для 16 элементов с компараторами [1:2][4:3][5:6][8:7] и т. д. В упр. 11 доказывается, что это сеть сортировки.

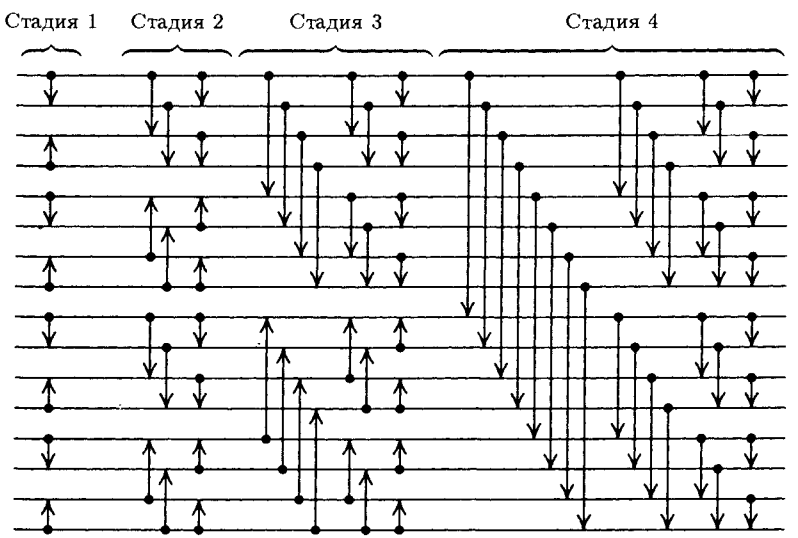


Рис. 56. Нестандартная сеть, основанная на битонной сортировке.

Если $x = \langle x_1, \ldots, x_n\rangle$ — это n-мерный вектор, а α — n-сеть, то обозначим через $x\alpha$ вектор чисел $\langle (x\alpha)_1, \ldots, (x\alpha)_n\rangle$, порожденных сетью. Положим также для краткости $a \vee b = \max(a,b)$, $a \wedge b = \min(a,b)$, $\bar{a} = 1-a$; тогда $(x[i:j])_i = x_i \wedge x_j$, $(x[i:j])_j = x_i \vee x_j$ и

$(x[i:j])_k = x_k$, когда $i \neq k \neq j$. Будем говорить, что α является *сетью сортировки*, если $(x\alpha)_i \leq (x\alpha)_{i+1}$ для всех x и для $1 \leq i < n$.

Символ $e^{(i)}$ обозначает вектор, в позиции i которого находится 1, а в остальных позициях — 0; таким образом, $(e^{(i)})_j = \delta_{ij}$. Символ D_n обозначает множество всех $2^n n$-мерных векторов из 0 и 1, а P_n — множество всех $n!$ векторов, являющихся перестановками $\{1, 2, \ldots, n\}$. Мы будем использовать обозначения $x \wedge y$ и $x \vee y$ для векторов $\langle x_1 \wedge y_1, \ldots, x_n \wedge y_n\rangle$ и $\langle x_1 \vee y_1, \ldots, x_n \vee y_n\rangle$ и будем писать $x \leq y$, если $x_i \leq y_i$ при всех i. Таким образом, $x \leq y$ тогда и только тогда, когда $x \vee y = y$, либо тогда и только тогда, когда $x \wedge y = x$. Если x и y лежат в D_n, то будем говорить, что x *покрывает* y, если $x = (y \vee e^{(i)}) \neq y$ при некотором i. Наконец, для всех x в D_n пусть $\nu(x)$ будет числом единиц в x, а $\zeta(x)$ — числом нулей; таким образом, $\nu(x) + \zeta(x) = n$.

1. [*20*] Нарисуйте диаграмму сети четно-нечетного слияния для $m = 3$ и $n = 5$.

2. [*22*] Покажите, что алгоритму сортировки В. Пратта (см. упр. 5.2.1–30) соответствует сеть сортировки n элементов, имеющая приблизительно $(\log_2 n)(\log_3 n)$ уровней задержки. Нарисуйте такую сеть для $n = 12$.

3. [*M20*] (К. Э. Бэтчер (K. E. Batcher).) Найдите простое соотношение между $C(m, m-1)$ и $C(m, m)$.

▶ **4.** [*M23*] Докажите, что $\hat{T}(6) = 5$.

5. [*M16*] Докажите, что выражение (13) действительно определяет время задержки для сети сортировки, описанной соотношениями (10).

6. [*28*] Пусть $T(n)$ — минимальное число стадий, требуемых для сортировки n различных чисел посредством *одновременного выполнения непересекающихся* сравнений (без соблюдения сетевого ограничения); каждое такое множество сравнений может быть представлено узлом, содержащим множество пар $\{i_1 : j_1, i_2 : j_2, \ldots, i_r : j_r\}$, где все $i_1, j_1, i_2, j_2, \ldots, i_r, j_r$ различны; от этого узла отходит вниз 2^r ветвей, соответствующих случаям

$$\langle K_{i_1} < K_{j_1},\ K_{i_2} < K_{j_2},\ \ldots,\ K_{i_r} < K_{j_r}\rangle,$$
$$\langle K_{i_1} > K_{j_1},\ K_{i_2} < K_{j_2},\ \ldots,\ K_{i_r} < K_{j_r}\rangle \qquad \text{и т. д.}$$

Докажите, что $T(5) = T(6) = 5$.

7. [*25*] Покажите, что если три последних компаратора сети для $n = 10$ на рис. 49 заменить "слабой" последовательностью [5:6][4:5][6:7], то сеть по-прежнему будет выполнять сортировку.

8. [*M20*] Докажите, что $\hat{M}(m_1+m_2, n_1+n_2) \geq \hat{M}(m_1, n_1) + \hat{M}(m_2, n_2) + \min(m_1, n_2)$ при $m_1, m_2, n_1, n_2 \geq 0$.

9. [*M25*] (Р. У. Флойд (R. W. Floyd).) Докажите, что $\hat{M}(3,3) = 6$, $\hat{M}(4,4) = 9$, $\hat{M}(5,5) = 13$.

10. [*M22*] Докажите, что битонный сортировщик Бэтчера, определенный в разделе, который предшествует (15), действительно работает. [*Указание.* Достаточно доказать, что будут сортироваться все последовательности, состоящие из k единиц, за которыми следуют l нулей, за которыми следуют $n - k - l$ единиц.]

11. [*M23*] Докажите, что битонный сортировщик Бэтчера порядка 2^t будет сортировать не только последовательности $\langle z_0, z_1, \ldots, z_{2^t-1}\rangle$, для которых $z_0 \geq \cdots \geq z_k \leq \cdots \leq z_{2^t-1}$, но и все последовательности, для которых $z_0 \leq \cdots \leq z_k \geq \cdots \geq z_{2^t-1}$. [Как следствие этого сеть на рис. 56 будет сортировать 16 элементов, так как каждая стадия состоит из битонных сортировщиков или обращенных битонных сортировщиков, применяемых к последовательностям, которые были рассортированы в противоположных направлениях.]

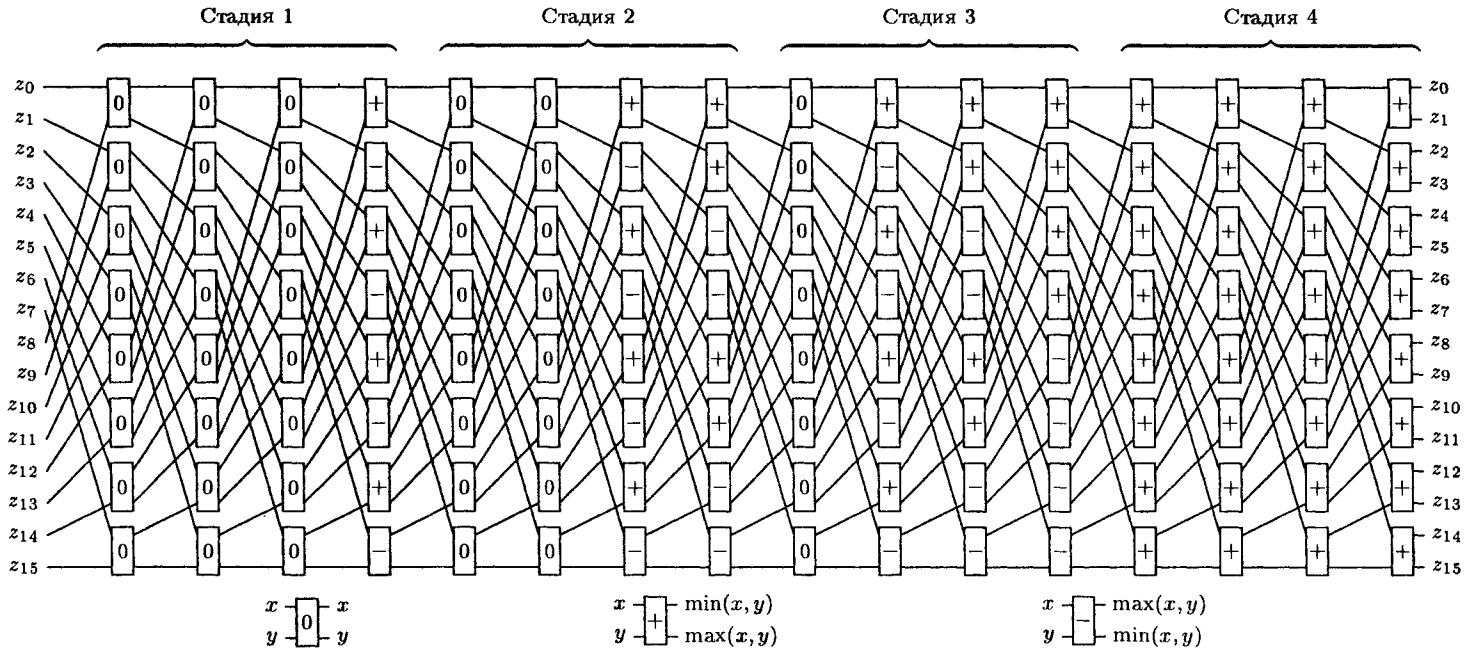


Рис. 57. Сортировка 16-ти идеально перемешанных элементов.

12. [*M20*] Докажите или опровергните следующее утверждение: если x и y — битонные последовательности равной длины, то последовательности $x \vee y$ и $x \wedge y$ также битонные.

▶ **13.** [*24*] (Х. С. Стоун (H. S. Stone).) Покажите, что сеть сортировки для 2^t элементов можно построить по схеме, представленной на рис. 57, для случая $t = 4$. Каждый из t^2 шагов этой схемы состоит из "идеального тасования" первых 2^{t-1} элементов с последними 2^{t-1}, за которым следуют операции, выполняемые одновременно над 2^{t-1} парами соседних элементов. Каждая из этих операций обозначена либо через "0" (нет операции), либо через "+" (стандартный модуль компаратора), либо через "−" (обращенный модуль компаратора). Сортировка протекает в t стадий по t шагов каждая; на последней стадии все операции суть "+". В течение стадии s при $s < t$ мы выполняем $t - s$ шагов, где все операции суть "0", а затем выполняем s шагов, где на каждом q-м шаге поочередно выполняются 2^{q-1} операций "+" и затем 2^{q-1} операций "−" при $q = 1, 2, \ldots, s$.

[Попутно отметим, что эта схема сортировки может быть реализована весьма простым устройством, реализующим один шаг "тасования и операций" и возвращающим выход на вход. Первые три шага на рис. 57 можно, конечно, исключить. Они оставлены лишь для того, чтобы сделать схему понятнее. Стоун замечает, что тот же принцип "тасования/операции" встречается в других алгоритмах, таких как быстрое преобразование Фурье (см. формулу 4.6.4-(40)).]

▶ **14.** [*M27*] (В. Е. Алексеев.) Пусть $\alpha = [i_1 : j_1] \ldots [i_r : j_r]$ представляет собой n-сеть; при $1 \le s \le r$ определим $\alpha^s = [i'_1 : j'_1] \ldots [i'_{s-1} : j'_{s-1}][i_s : j_s] \ldots [i_r : j_r]$, где i'_k и j'_k получены из i_k и j_k путем замены i_s элементом j_s и замены j_s элементом i_s во всех случаях, когда они встречаются. Например, если $\alpha = [1:2][3:4][1:3][2:4][2:3]$, то $\alpha^4 = [1:4][3:2][1:3][2:4][2:3]$.

a) Докажите, что $D_n\alpha = D_n(\alpha^s)$.

b) Докажите, что $(\alpha^s)^t = (\alpha^t)^s$.

c) *Сопряженной* с α является любая сеть вида $(\ldots((\alpha^{s_1})^{s_2})\ldots)^{s_k}$. Докажите, что α имеет не более 2^{r-1} сопряженных сетей.

d) Пусть $g_\alpha(x) = [x \in D_n\alpha]$ и пусть $f_\alpha(x) = (\bar{x}_{i_1} \vee x_{j_1}) \wedge \cdots \wedge (\bar{x}_{i_r} \vee x_{j_r})$. Докажите, что $g_\alpha(x) = \bigvee\{f_{\alpha'}(x) \mid \alpha'$ является сопряженной с $\alpha\}$.

e) Пусть G_α есть ориентированный граф с вершинами $\{1, \ldots, n\}$ и дугами $i_s \to j_s$ при $1 \le s \le r$. Докажите, что α является сетью сортировки тогда и только тогда, когда $G_{\alpha'}$ имеет направленный путь от i к $i+1$ при $1 \le i < n$ и для всех α', сопряженных с α. [Это условие весьма примечательно, поскольку G_α не зависит от порядка компараторов в α.]

15. [*20*] Найдите нестандартную сеть сортировки четырех элементов, содержащую только пять модулей компараторов.

16. [*M22*] Докажите, что следующий алгоритм преобразует любую сеть сортировки $[i_1 : j_1] \ldots [i_r : j_r]$ в стандартную сеть сортировки той же длины.

T1. Пусть q — наименьший индекс, такой, что $i_q > j_q$. Если таких индексов нет, то остановиться.

T2. Заменить все вхождения i_q элементами j_q и все вхождения j_q элементами i_q во всех компараторах $[i_s : j_s]$ для $q \le s \le r$. Вернуться к шагу T1. ▌

Так, $[4:1][3:2][1:3][2:4][1:2][3:4]$ преобразуется сначала в $[1:4][3:2][4:3][2:1][4:2][3:1]$, затем в $[1:4][2:3][4:2][3:1][4:3][2:1]$, затем в $[1:4][2:3][2:4][3:1][2:3][4:1]$ и т. д., пока не получится стандартная сеть $[1:4][2:3][2:4][1:3][1:2][3:4]$.

17. [*M25*] Пусть D_{tn} — множество всех $\binom{n}{t}$ последовательностей $\langle x_1, \ldots, x_n \rangle$ из нулей и единиц, имеющих ровно t единиц. Докажите, что $\hat{U}_t(n)$ равно минимальному числу компараторов, которые необходимы в сети, сортирующей все элементы D_{tn}; что $\hat{V}_t(n)$ равно минимальному числу компараторов, необходимых для сортировки $D_{tn} \cup D_{(t-1)n}$; и что $\hat{W}_t(n)$ равно минимальному числу компараторов, необходимых для сортировки $\bigcup_{0 \le k \le t} D_{kn}$.

▶ **18.** [*M20*] Докажите, что для сети, которая определяет медиану $2t-1$ элементов, требуется не менее $(t-1)\lceil\lg(t+1)\rceil+\lceil\lg t\rceil$ модулей компараторов. [*Указание.* См. доказательство теоремы А.]

19. [*M22*] Докажите, что $\hat{U}_2(n)=2n-4$ и $\hat{V}_2(n)=2n-3$ для всех $n\geq 2$.

20. [*24*] Докажите, что $\hat{V}_3(5)=7$.

21. [*21*] Подтвердите или опровергните следующее утверждение: вставка нового стандартного компаратора в любую стандартную сеть сортировки приведет к образованию новой стандартной сети сортировки.

22. [*M17*] Пусть α — любая n-сеть, а x и y — два n-вектора.

a) Докажите, что из $x\leq y$ следует $x\alpha\leq y\alpha$.

b) Докажите, что $x\cdot y\leq(x\alpha)\cdot(y\alpha)$, где $x\cdot y$ означает скалярное произведение $x_1y_1+\cdots+x_ny_n$.

23. [*M18*] Пусть α есть n-сеть. Докажите, что существует перестановка $p\in P_n$, такая, что $(p\alpha)_i=j$ тогда и только тогда, когда в D_n найдутся векторы x и y, такие, что x покрывает y, $(x\alpha)_i=1$, $(y\alpha)_i=0$ и $\zeta(y)=j$.

▶ **24.** [*M21*] (В. Е. Алексеев.) Пусть α есть n-сеть; введем обозначения

$$l_k=\min\{(p\alpha)_k\mid p\in P_n\},\qquad u_k=\max\{(p\alpha)_k\mid p\in P_n\}$$

при $1\leq k\leq n$ для нижней и верхней границ диапазона значений, которые могут появляться на линии выхода k. Пусть l'_k и u'_k — аналогично определенные величины для сети $\alpha'=\alpha[i:j]$. Докажите, что

$$l'_i=l_i\wedge l_j,\qquad l'_j\leq l_i+l_j,\qquad u'_i\geq u_i+u_j-(n+1),\qquad u'_j=u_i\vee u_j.$$

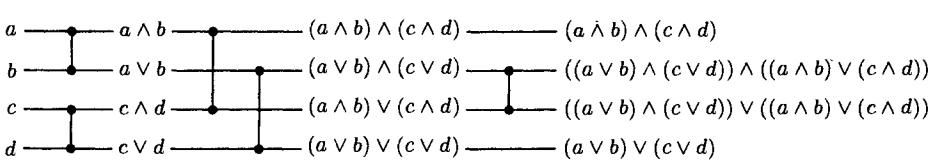
[*Указание.* Для данных векторов x и y из D_n, таких, что $(x\alpha)_i=(y\alpha)_j=0$, $\zeta(x)=l_i$ и $\zeta(y)=l_j$, найдите вектор z из D_n, такой, что $(z\alpha')_j=0$, $\zeta(z)\leq l_i+l_j$.]

25. [*M30*] Пусть l_k и u_k определены, как в упр. 24. Докажите, что множество $\{(p\alpha)_k\mid p$ in $P_n\}$ содержит все целые числа между l_k и u_k включительно.

26. [*M24*] (Р. У. Флойд.) Пусть α есть n-сеть. Докажите, что множество $D_n\alpha=\{x\alpha\mid x$ in $D_n\}$ может быть определено из множества $P_n\alpha=\{p\alpha\mid p$ in $P_n\}$ и, обратно, $P_n\alpha$ может быть определено из $D_n\alpha$.

▶ **27.** [*M20*] Пусть x и y — векторы и пусть $x\alpha$ и $y\alpha$ — рассортированные векторы. Докажите, что $(x\alpha)_i\leq(y\alpha)_j$ тогда и только тогда, когда для любой совокупности j элементов из y можно найти совокупность i элементов из x, такую, что любой элемент, взятый из x, меньше некоторого элемента, взятого из y, или равен ему. Используйте этот принцип для доказательства того, что *если рассортировать строки любой матрицы, а затем рассортировать столбцы, то строки останутся упорядоченными.*

▶ **28.** [*M20*] На следующей диаграмме показано, как записать формулы для содержимого всех линий сети сортировки через ее входы.



Используя законы коммутативности $x\wedge y=y\wedge x$, $x\vee y=y\vee x$, законы ассоциативности $x\wedge(y\wedge z)=(x\wedge y)\wedge z$, $x\vee(y\vee z)=(x\vee y)\vee z$, законы дистрибутивности $x\wedge(y\vee z)=(x\wedge y)\vee(x\wedge z)$, $x\vee(y\wedge z)=(x\vee y)\wedge(x\vee z)$, законы поглощения $x\wedge(x\vee y)=x\vee(x\wedge y)=x$

и законы идемпотентности $x \wedge x = x \vee x = x$, мы можем свести формулы в правой части этой сети соответственно к $(a \wedge b \wedge c \wedge d)$, $(a \wedge b \wedge c) \vee (a \wedge b \wedge d) \vee (a \wedge c \wedge d) \vee (b \wedge c \wedge d)$, $(a \wedge b) \vee (a \wedge c) \vee (a \wedge d) \vee (b \wedge c) \vee (b \wedge d) \vee (c \wedge d)$ и $a \vee b \vee c \vee d$.

Докажите, что в общем случае t-й в порядке убывания элемент из $\{x_1, \ldots, x_n\}$ задается "элементарной симметрической функцией"

$$\sigma_t(x_1, \ldots, x_n) = \bigvee \{x_{i_1} \wedge x_{i_2} \wedge \cdots \wedge x_{i_t} \mid 1 \le i_1 < i_2 < \cdots < i_t \le n\}.$$

[Здесь $\binom{n}{t}$ членов объединяются с помощью операции $\vee$. Таким образом, задача нахождения сети сортировки минимальной стоимости эквивалентна задаче вычисления элементарных симметрических функций с минимальным числом схем "и/или", где на каждом шаге величины ϕ и ψ заменяются величинами $\phi \wedge \psi$ и $\phi \vee \psi$.]

29. [*M20*] Дано, что $x_1 \le x_2 \le x_3$ и $y_1 \le y_2 \le y_3 \le y_4 \le y_5$ и что $z_1 \le z_2 \le \cdots \le z_8$ — результат слияния x с y. Найдите выражения для каждого z_k в терминах x_k и y_k, используя операторы $\wedge$ и $\vee$.

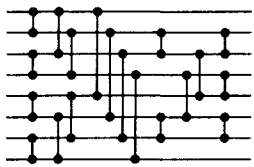
30. [*HM24*] Докажите, что любая формула, содержащая $\wedge$ и $\vee$ и независимые переменные $\{x_1, \ldots, x_n\}$, может быть приведена с использованием тождеств из упр. 28 к каноническому виду $\tau_1 \vee \tau_2 \vee \cdots \vee \tau_k$. Здесь $k \ge 1$ и каждый τ_i имеет вид $\bigwedge\{x_j \mid j \text{ in } S_i\}$, где S_i — подмножество $\{1, 2, \ldots, n\}$ и никакое множество S_i не включается в S_j, если $i \ne j$. Докажите также, что две такие канонические формы равны для всех $x_1, \ldots, x_n$ тогда и только тогда, когда они идентичны (с точностью до порядка).

31. [*M24*] (Р. Дедекинд (R. Dedekind), 1897.) Пусть δ_n — число различных канонических форм от $x_1, \ldots, x_n$ в смысле упр. 30. Так, $\delta_1 = 1$, $\delta_2 = 4$ и $\delta_3 = 18$. Чему равно δ_4?

32. [*M28*] (М. У. Грин (M. W. Green).) Пусть $G_1 = \{00, 01, 11\}$; определим G_{t+1} как множество всех цепочек $\theta\phi\psi\omega$, таких, что θ, ϕ, ψ, ω имеют длину 2^{t-1} и $\theta\phi$, $\psi\omega$, $\theta\psi$ и $\phi\omega$ принадлежат G_t. Пусть α — сеть, состоящая из четырех первых уровней 16-элементного сортировщика, изображенного на рис. 49. Покажите, что $D_{16}\alpha = G_4$, и докажите, что это множество имеет в точности $\delta_4 + 2$ элементов (см. упр. 31).

▶ **33.** [*M22*] Не все δ_n функций от $\langle x_1, \ldots, x_n \rangle$ из упр. 31 могут встретиться в сетях компараторов. Докажите, что функция $(x_1 \wedge x_2) \vee (x_2 \wedge x_3) \vee (x_3 \wedge x_4)$ не может быть результатом работы никакой сети компараторов от $\langle x_1, \ldots, x_n \rangle$.

34. [*23*] Является ли следующая сеть сетью сортировки?



35. [*20*] Докажите, что в любой стандартной сети сортировки должен, по крайней мере, один раз встретиться каждый из *соседних* компараторов $[i{:}i{+}1]$ при $1 \le i < n$.

▶ **36.** [*22*] Сеть, представленная на рис. 47, содержит только кратчайшие сравнения $[i{:}i{+}1]$. Будем называть такие сети *примитивными*.

a) Докажите, что примитивная сеть сортировки для n элементов должна иметь не менее $\binom{n}{2}$ компараторов. [*Указание.* Рассмотрите инверсии перестановки.]

b) (Р. У. Флойд, 1964.) Пусть α — примитивная сеть для n элементов, а x — вектор, такой, что $(x\alpha)_i > (x\alpha)_j$ при некоторых $i < j$. Докажите, что $(y\alpha)_i > (y\alpha)_j$, где y — вектор $\langle n, n{-}1, \ldots, 1 \rangle$.

c) В качестве следствия (b) докажите, что примитивная сеть является сетью сортировки тогда и только тогда, когда она сортирует единственный вектор $\langle n, n{-}1, \ldots, 1 \rangle$.

37. [*M22*] *Четно-нечетная сортировка* с транспозициями для n чисел, $n \geq 3$, — это n-уровневая сеть с $\frac{1}{2}n(n-1)$ компараторами, напоминающая кирпичную кладку (рис. 58). (Если n четно, имеется два варианта.) Такую сортировку особенно легко реализовать аппаратно, поскольку выполняются попеременно только два вида действий. Докажите, что такая сеть действительно будет работающей сетью сортировки. [*Указание.* См. упр. 36.]

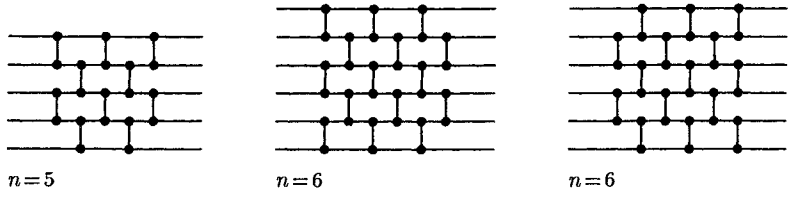


Рис. 58. Четно-нечетная сортировка с транспозициями.

▶ **38.** [*43*] Пусть $N = \binom{n}{2}$. Найдите взаимно однозначное соответствие между диаграммами Юнга вида $(n-1, n-2, \ldots, 1)$ и примитивными сетями сортировки $[i_1 : i_1+1] \ldots [i_N : i_N+1]$. [Следовательно, по теореме 5.1.4Н существует точно

$$\frac{N!}{1^{n-1}\, 3^{n-2}\, 5^{n-3} \ldots (2n-3)^1}$$

таких сетей сортировки.] *Указание.* В упр. 36, (с) показано, что примитивные сети без избыточных компараторов соответствуют пути от $1\,2 \ldots n$ до $n \ldots 2\,1$ на многограннике, подобном изображенному на рис. 1 в разделе 5.1.1.

39. [*25*] Пусть известно, что примитивная сеть компараторов состоит из n линий и правильно сортирует единственный вход $1\,0\,1\,0 \ldots 1\,0$. (См. упр. 36; полагается, что n четное.) Покажите, что в такой сети "средняя треть", состоящая из всех компараторов, подключенных только к линиям от $\lceil n/3 \rceil$ до $\lceil 2n/3 \rceil$ включительно, будет сортировать *любые* входы.

40. [*HM44*] Пусть компараторы $[i_1 : i_1+1][i_2 : i_2+1] \ldots [i_r : i_r+1]$ выбираются случайно, причем каждое значение $i_k \in \{1, 2, \ldots, n-1\}$ равновероятно. Процесс прекращается, когда в составе сети в качестве подсети появляется конфигурация сортировки по методу пузырька, подобная изображенной на рис. 47. Докажите, что $r \leq 4n^2 + O(n^{3/2} \log n)$, а вероятность остальных случаев — $O(n^{-1000})$.

41. [*M47*] Пусть компараторы $[i_1 : j_1][i_2 : j_2] \ldots [i_r : j_r]$ выбираются случайным образом, причем каждый *неизбыточный* выбор $1 \leq i_k < j_k \leq n$ имеет равную вероятность. Процесс остановится, когда получится сеть сортировки. Оцените ожидаемое значение r; будет ли оно $O(n^{1+\epsilon})$ при всех $\epsilon > 0$?

▶ **42.** [*25*] (Д. Ван Ворис (D. Van Voorhis).) Докажите, что $\hat{S}(n) \geq \hat{S}(n-1) + \lceil \lg n \rceil$.

43. [*48*] Найдите (m, n)-сеть слияния с числом компараторов, меньшим $C(m, n)$, или докажите, что такой сети не существует.

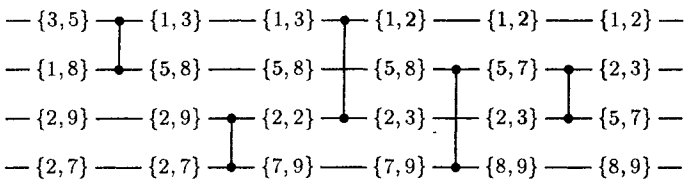
44. [*50*] Найдите точное значение $\hat{S}(n)$ для какого-либо $n > 8$.

45. [*M20*] Докажите, что любая $(1, n)$-сеть сортировки без разветвлений должна иметь по крайней мере $\lceil \lg(n+1) \rceil$ уровней задержки.

▶ **46.** [*30*] (М. Айгнер (M. Aigner).) Покажите, что при использовании любого алгоритма, который способен параллельно выполнять несвязанные сравнения минимальное число стадий, необходимых для слияния m элементов, будет не меньше $\lceil \lg(m+n) \rceil$; следовательно, битонная сеть сортировки имеет оптимальную задержку.

47. [*47*] Является ли функция $T(n)$ в упр. 6 строго меньшей, чем $\hat{T}(n)$ при некотором n?

▶ **48.** [*26*] Можно дать другую интерпретацию сетей сортировки, считая, что на каждой линии находится не одно число, а мультимножество из m чисел; при такой интерпретации операция $[i:j]$ заменяет x_i и x_j соответственно значениями $x_i \mathbin{\Cap} x_j$ и $x_i \mathbin{\Cup} x_j$ — наименьшими m и наибольшими m из $2m$ чисел $x_i \uplus x_j$. (Например, на приведенной ниже схеме иллюстрируется такая интерпретация при $m = 2$; каждый компаратор сливает свои входы и отделяет нижнюю половину от верхней.)



Если a и b суть мультимножества, содержащие по m чисел, то будем говорить, что $a \ll b$ тогда и только тогда, когда $a \mathbin{\Cap} b = a$ (или, что эквивалентно, $a \mathbin{\Cup} b = b$; наибольший элемент a меньше или равен наименьшему элементу b). Таким образом, $a \mathbin{\Cap} b \ll a \mathbin{\Cup} b$.

Пусть α есть n-сеть, а $x = (x_1, \ldots, x_n)$ — вектор, в котором каждая компонента x_i — мультимножество из m элементов. Докажите, что если $(x\alpha)_i$ не $\ll$ $(x\alpha)_j$ в описанной интерпретации, то в D_n найдется вектор y, такой, что $(y\alpha)_i = 1$ и $(y\alpha)_j = 0$. [Следовательно, сеть сортировки n элементов превращается в сеть сортировки mn элементов, если заменить сравнения m-путевыми слияниями. На рис. 59 изображен 8-элементный сортировщик, построенный из 4-элементного с использованием этого вывода.]

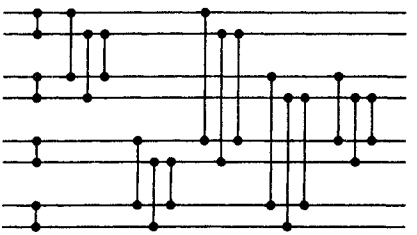


Рис. 59. 8-элементный сортировщик, построенный из 4-элементного сортировщика с использованием слияния.

49. [*M23*] Покажите, что в обозначениях упр. 48 $(x \mathbin{\Cap} y) \mathbin{\Cap} z = x \mathbin{\Cap} (y \mathbin{\Cap} z)$ и $(x \mathbin{\Cup} y) \mathbin{\Cup} z = x \mathbin{\Cup} (y \mathbin{\Cup} z)$; однако $(x \mathbin{\Cup} y) \mathbin{\Cap} z$ *не всегда* равно $(x \mathbin{\Cap} z) \mathbin{\Cup} (y \mathbin{\Cap} z)$ и $(x \mathbin{\Cap} y) \mathbin{\Cup} (x \mathbin{\Cap} z) \mathbin{\Cup} (y \mathbin{\Cap} z)$ *не всегда* равно средним m элементам $x \uplus y \uplus z$. Найдите правильную формулу для этих средних элементов, использовав в ней x, y, z, а также операции $\mathbin{\Cap}$ и $\mathbin{\Cup}$.

50. [*HM46*] Исследуйте свойства операций $\mathbin{\Cap}$ и $\mathbin{\Cup}$, определенных в упр. 48. Можно ли охарактеризовать все тождества в этой алгебре каким-либо изящным способом или вывести все их из конечного набора тождеств? В этом отношении такие тождества, как $x \mathbin{\Cap} x \mathbin{\Cap} x = x \mathbin{\Cap} x$ и $x \mathbin{\Cap} (x \mathbin{\Cup} (x \mathbin{\Cap} (x \mathbin{\Cup} y))) = x \mathbin{\Cap} (x \mathbin{\Cup} y)$, которые имеют место только для $m \le 2$, представляют относительно небольшой интерес; рассматривайте лишь тождества, справедливые при всех m.

▶ **51.** [*M25*] (Р. Л. Грэхем (R. L. Graham).) Компаратор $[i:j]$ называется избыточным в сети $\alpha_1[i:j]\alpha_2$, если либо $(x\alpha_1)_i \le (x\alpha_1)_j$ для всех векторов x, либо $(x\alpha_1)_i \ge (x\alpha_1)_j$ для всех векторов x. Докажите, что если α является сетью с r неизбыточными компараторами, то найдется по крайней мере r различных упорядоченных пар (i, j) различных индексов, таких, что $(x\alpha)_i \le (x\alpha)_j$ для всех векторов x. (Следовательно, сеть без избыточных компараторов содержит не более $\binom{n}{2}$ модулей.)

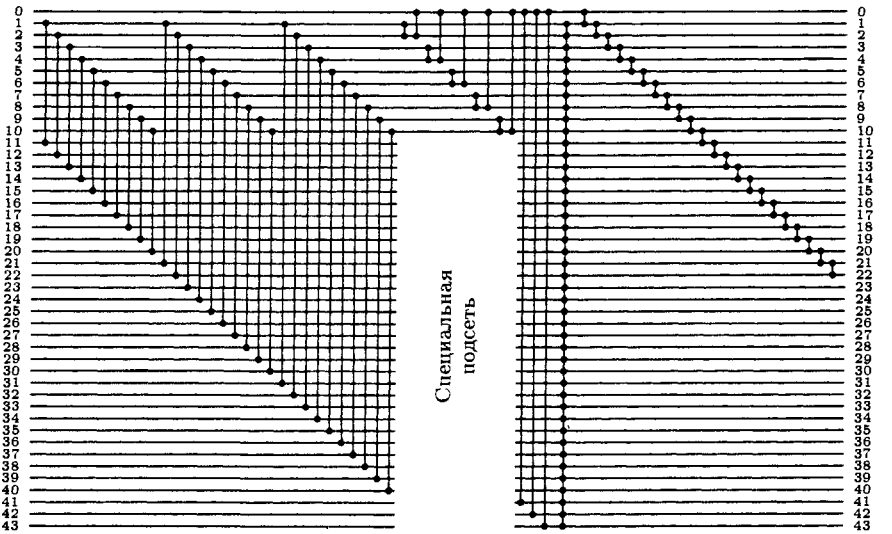


Рис. 60. Семейство сетей, возможность которых выполнять сортировку очень сложно проверить. Показан вариант при $m = 3$ и $n = 5$ (см. упр. 52).

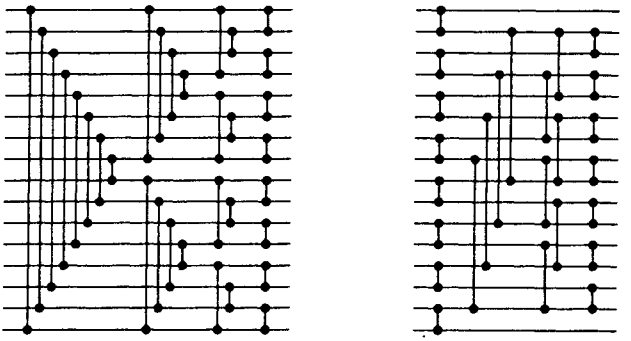
▶ **52.** [*32*] (М. О. Рабин (M. O. Rabin), 1980.) Докажите, что в общем случае исключительно трудно дать заключение, является ли данная последовательность компараторов сетью сортировки, анализируя сеть, аналогичную представленной на рис. 60. Принято перенумеровывать входы x_0 до x_N, где $N = 2mn + m + 2n$; положительные целые числа являются параметрами. Первые компараторы — $[j:j+2nk]$ при $1 \le j \le 2n$ и $1 \le k \le m$. Тогда имеем $[2j-1:2j][0:2j]$ при $1 \le j \le n$ параллельно со специальной подсетью, которая использует только индексы $> 2n$. Далее сравниваем $[0:2mn+2n+j]$ при $1 \le j \le m$. И наконец, существует законченная сеть сортировки для $\langle x_1, \ldots, x_N \rangle$, за которой следует $[0:1][1:2]\ldots[N-t-1:N-t]$, где $t = mn + n + 1$.

a) Опишите все входы $\langle x_0, x_1, \ldots, x_N \rangle$, которые *не* сортируются такой сетью, в терминах поведения специальной подсети.

b) Задав множество выражений наподобие $(y_1 \vee y_2 \vee \bar{y}_3) \wedge (\bar{y}_2 \vee y_3 \vee \bar{y}_4) \wedge \ldots$, объясните, как построить специальную подсеть по типу сети, показанной на рис. 60, которая сортировала бы все входы тогда и только тогда, когда выражение не удовлетворяется.

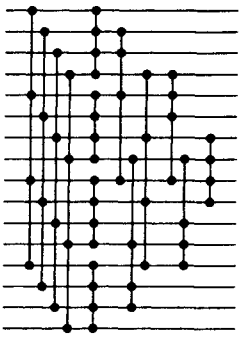
53. [*30*] (*Периодическая сеть сортировки.*) Две показанные ниже сети следует считать иллюстрацией рекурсивного построения t-уровневой сети при $n = 2^t$ в случае $t = 4$. Если пронумеровать линии входов от 0 до $2^t - 1$, то l-й уровень в случае (a) имеет компараторы $[i:j]$, где $i \bmod 2^{t+1-l} < 2^{t-l}$ и $j = i \oplus (2^{t+1-l} - 1)$; всего существует $t2^{t-1}$ компараторов, как и в сети битонного слияния. В случае (b) компараторы первого уровня суть $[2j:2j+1]$ при $0 \le j < 2^{t-1}$ и l-уровневые компараторы при $2 \le l \le t$ суть $[2j+1:2j+2^{t+1-l}]$ при $0 \le j < 2^{t-1} - 2^{t-l}$; всего имеется $(t-1)2^{t-1}+1$ компараторов, как в сети четно-нечетного слияния.

Докажите, что если входные числа 2^k-упорядочены, как в теореме 5.2.1H при некотором $k \ge 1$, то обе сети приведут к выходу, который будет 2^{k-1}-упорядочен. Таким образом, мы сможем сортировать 2^t чисел, пропуская их через любую из сетей t раз. [Когда t велико, такие сети сортировки выполняют примерно вдвое больше сравнений, чем алгоритм 5.2.2M; но общая временна́я задержка та же, что и на рис. 57, а реализация выполняется проще, поскольку та же самая сеть используется многократно.]



(a) (b)

54. [*42*] Проанализируйте свойства сетей сортировки, построенных не из 2-элементных сортировщиков, а из m-сортирующих модулей. (Например, Дж. Шапиро (G. Shapiro) построил сеть



для сортировки 16 элементов, использовав четырнадцать 4-элементных сортировщиков. Наилучшее ли это решение? Докажите, что m^2 элементов можно рассортировать с помощью не более 16 уровней m-элементных сортировщиков, если m достаточно велико.)

55. [*23*] *Перестановочной сетью* называется последовательность модулей $[i_1 : j_1] \ldots [i_r : j_r]$, где каждый модуль $[i : j]$ может устанавливаться извне в одно из двух состояний: либо он передает свои входы без изменений, либо меняет местами x_i и x_j (независимо от значений x_i и x_j). Последовательность модулей должна быть такой, чтобы на выходе можно было получить любую перестановку входов при соответствующей установке модулей. Любая сеть сортировки является, очевидно, перестановочной сетью, но обратное неверно. Найдите перестановочную сеть для пяти элементов, имеющую только восемь модулей.

▶ **56.** [*25*] Предположим, что битовый вектор $x \in D_n$ не рассортирован. Покажите, что существует стандартная n-сеть α_x, которая не сможет рассортировать x, хотя она и сортирует другие элементы D_n.

57. [*M35*] *Четно-нечетное слияние* аналогично четно-четному слиянию Бэтчера, но, если $mn > 2$, процесс рекурсивно сливает последовательность $\langle x_{m \bmod 2+1}, \ldots, x_{m-3}, x_{m-1} \rangle$ с $\langle y_1, y_3, \ldots, y_{2\lceil n/2 \rceil - 1} \rangle$ и $\langle x_{(m+1) \bmod 2+1}, \ldots, x_{m-2}, x_m \rangle$ с $\langle y_2, y_4, \ldots, y_{2\lfloor n/2 \rfloor} \rangle$ прежде, чем будет сформировано множество $\lceil m/2 \rceil + \lceil n/2 \rceil - 1$ сравнений-обменов, аналогичных (1). Покажите, что при четно-нечетном слиянии достигается оптимальное для битонного слияния время задержки $\lceil \lg(m+n) \rceil$, поскольку не делается сравнений больше, чем в методе битонного слияния. Фактически нужно доказать, что число сравнений $A(m,n)$, выполняемых при четно-нечетном слиянии, удовлетворяет условию $C(m,n) \le A(m,n) < \frac{1}{2}(m+n) \lg \min(m,n) + m + \frac{3}{2}n$.

УПРАЖНЕНИЯ (часть 2)

Следующие упражнения имеют отношение к различным аспектам оптимизации, касающимся сортировки. Несколько первых упражнений основаны на интересном "многоголовочном" обобщении метода пузырька, предложенном Ф. Н. Армстронгом (P. N. Armstrong) и Р. Дж. Нельсоном (R. J. Nelson) еще в 1954 году. [См. *U.S. Patents 3029413, 3034102*.] Пусть $1 = h_1 < h_2 < \cdots < h_m = n$ — возрастающая последовательность целых чисел; будем называть ее последовательностью голов длиной m с диапазоном n. Она будет использоваться при определении методов сортировки специального вида. Сортировка записей $R_1 \ldots R_N$ осуществляется за несколько проходов, а каждый проход состоит из $N + n - 1$ шагов. На шаге j при $j = 1 - n,\ 2 - n,\ \ldots,\ N - 1$ рассматриваются записи $R_{j+h[1]}, R_{j+h[2]}, \ldots, R_{j+h[m]}$ и в случае необходимости переставляются так, чтобы их ключи оказались упорядоченными. (Мы говорим, что $R_{j+h[1]}, \ldots, R_{j+h[m]}$ находятся "под головками чтения/записи". Если $j + h[k]$ либо < 1, либо $> N$, то запись $R_{j+h[k]}$ не рассматривается. Иначе говоря, ключи $K_0, K_{-1}, K_{-2}, \ldots$ считаются равными $-\infty$, а $K_{N+1}, K_{N+2}, \ldots$ считаются равными $+\infty$. Поэтому при $j \le -h[m-1]$ или $j > N - h[2]$ шаг j становится тривиальным.)

Например, ниже показан один проход сортировки при $m = 3$, $N = 9$ и $h_1 = 1$, $h_2 = 2$, $h_3 = 4$.

	K_{-2}	K_{-1}	K_0	K_1	K_2	K_3	K_4	K_5	K_6	K_7	K_8	K_9	K_{10}	K_{11}	K_{12}
$j = -3$	—	—		3	1	4	5	9	2	6	8	7			
$j = -2$		—	—	3	1	4	5	9	2	6	8	7			
$j = -1$			—	3	1	4	5	9	2	6	8	7			
$j = 0$				1	3	4	5	9	2	6	8	7			
$j = 1$				1	3	4	5	9	2	6	8	7			
$j = 2$				1	3	2	4	9	5	6	8	7			
$j = 3$				1	3	2	4	6	5	9	8	7			
$j = 4$				1	3	2	4	5	6	9	8	7			
$j = 5$				1	3	2	4	5	6	7	8	9			
$j = 6$				1	3	2	4	5	6	7	8	9	—		
$j = 7$				1	3	2	4	5	6	7	8	9		—	
$j = 8$				1	3	2	4	5	6	7	8	9	—		—

Если $m = 2$, $h_1 = 1$ и $h_2 = 2$, этот "многоголовочный" метод сводится к методу пузырька (алгоритм 5.2.2B).

58. [*21*] (Джеймс Дугунджи (James Dugundji).) Докажите, что если $h[k+1] = h[k]+1$ при некотором k, $1 \le k < m$, то многоголовочный сортировщик, описанный выше, рассортирует любой входной файл за конечное число проходов. Но если $h[k+1] \ge h[k]+2$ при $1 \le k < m$, то не исключен вариант, когда входная последовательность никогда не будет упорядочена.

▸ **59.** [*30*] (Армстронг (Armstrong) и Нельсон (Nelson).) Пусть $h[k+1] \le h[k] + k$ при $1 \le k < m$ и $N \ge n - 1$. Докажите, что в течение первого прохода наибольшие $n - 1$ элементов всегда займут свои окончательные места. [*Указание.* Используйте принцип нулей и единиц; докажите, что если сортируется последовательность из нулей и единиц, причем единиц меньше n, то все головки могут читать единицы лишь в том случае, когда все нули лежат слева от головок.] Докажите, что если головки удовлетворяют сформулированным условиям, то сортировка будет закончена не более чем за $\lceil (N-1)/(n-1) \rceil$ проходов. Существует ли входной файл, для которого необходимо так много проходов?

60. [*26*] Докажите, что, если $n = N$, при первом проходе наименьший ключ будет помещен в позицию R_1 тогда и только тогда, когда $h[k+1] \le 2h[k]$ при $1 \le k < m$.

61. [*34*] (Дж. Хопкрофт (J. Hopcroft).) "Совершенным сортировщиком" N элементов называется многоголовочный сортировщик, который всегда заканчивает работу за один проход. В упр. 59 доказано, что последовательность $\langle h_1, h_2, h_3, h_4, \ldots, h_m\rangle = \langle 1, 2, 4, 7, \ldots, 1 + \binom{m}{2}\rangle$ образует совершенный сортировщик для $N = \binom{m}{2} + 1$ элементов, используя $m = (\sqrt{8N-7}+1)/2$ головок. Например, последовательность головок $\langle 1, 2, 4, 7, 11, 16, 22\rangle$ является совершенным сортировщиком для 22 элементов.

Докажите, что последовательность головок $\langle 1, 2, 4, 7, 11, 16, 23\rangle$ на самом деле будет совершенным сортировщиком для 23 элементов.

62. [*49*] Определите наибольшее N, для которого существует совершенный сортировщик с m головками. Верно ли, что $N = O(m^2)$?

63. [*23*] (В. Пратт (V. Pratt).) Если каждая головка h_k находится в положении 2^{k-1}, $1 \le k \le m$, то сколько проходов потребуется для сортировки последовательности нулей и единиц $z_1 z_2 \ldots z_{2^m-1}$, где $z_j = 0$ тогда и только тогда, когда j является степенью 2?

64. [*24*] (*Однородная сортировка.*) В дереве, представленном на рис. 34 в разделе 5.3.1, сравнение 2:3 выполняется в обеих ветвях уровня 1, а в каждой ветви уровня 2 выполняется сравнение 1:3, если только оно не является избыточным. В общем случае мы можем рассмотреть класс алгоритмов сортировки, однородных именно в этом смысле. Предполагая, что $M = \binom{N}{2}$ пар $\{(a, b) \mid 1 \le a < b \le N\}$ выстроены в последовательность

$$(a_1, b_1), (a_2, b_2), \ldots, (a_M, b_M),$$

мы можем последовательно выполнять те сравнения $K_{a_1} : K_{b_1}$, $K_{a_2} : K_{b_2}$, ..., результат которых еще не известен. Каждая из $M!$ расстановок пар (a, b) определяет алгоритм однородной сортировки. Идея однородной сортировки принадлежит Х. Л. Бьюсу (H. L. Beus) [*JACM* **17** (1970), 482–495], в работе которого было предложено несколько следующих упражнений.

Для формального определения однородной сортировки удобно воспользоваться теорией графов. Пусть G — ориентированный граф с вершинами $\{1, 2, \ldots, N\}$ и без дуг. Для $i = 1, 2, \ldots, M$ мы добавляем дуги к G следующим образом.

Случай 1. В G имеется путь от a_i к b_i. Добавить к G дугу $a_i \to b_i$.

Случай 2. В G имеется путь от b_i к a_i. Добавить к G дугу $b_i \to a_i$.

Случай 3. В G нет пути ни от a_i к b_i, ни от b_i к a_i. Сравнить $K_{a_i} : K_{b_i}$; затем, если $K_{a_i} \le K_{b_i}$, добавить к G дугу $a_i \to b_i$; если же $K_{a_i} > K_{b_i}$, то добавить дугу $b_i \to a_i$.

Нас интересует, главным образом, число сравнений ключей, выполняемых алгоритмом однородной сортировки, а не механизм, с помощью которого действительно устраняются избыточные сравнения; граф G необязательно строить в явном виде — здесь он используется только для определения однородной сортировки.

Будем также рассматривать *ограниченную однородную сортировку*, при которой в указанных выше случаях 1–3 учитываются только пути длиной 2. (Алгоритм ограниченной сортировки может выполнять некоторые избыточные сравнения, но, как показано в упр. 65, анализ ограниченного случая выполняется несколько проще.)

Докажите, что алгоритм ограниченной однородной сортировки совпадает с алгоритмом однородной сортировки, когда последовательность пар лексикографически упорядочена:

$$(1,2)(1,3)(1,4)\ldots(1,N)(2,3)(2,4)\ldots(2,N)\ldots(N-1,N).$$

Покажите, что на самом деле оба алгоритма эквивалентны алгоритму быстрой сортировки (алгоритм 5.2.2Q), если все ключи различны и избыточные сравнения быстрой сортировки устранены, как в упр. 5.2.2–24. (Не обращайте внимания на порядок, в котором действи-

тельно выполняются сравнения при быстрой сортировке; учитывайте только то, какие пары ключей сравниваются.)

65. [*M38*] Для заданной, как в упр. 58, последовательности пар $(a_1, b_1) \ldots (a_M, b_M)$ пусть c_i будет числом пар (j, k), таких, что $j < k < i$ и (a_i, b_i), (a_j, b_j), (a_k, b_k) образуют треугольник.

a) Докажите, что среднее число сравнений, выполняемых алгоритмом ограниченной однородной сортировки, равно $\sum_{i=1}^{M} 2/(c_i + 2)$.

b) Используйте результат (a) и упр. 64, чтобы определить среднее число неизбыточных сравнений, выполняемых при быстрой сортировке.

c) Следующая последовательность пар навеяна сортировкой методом слияния (но не эквивалентна ей):

$$(1,2)(3,4)(5,6)\ldots(1,3)(1,4)(2,3)(2,4)(5,7)\ldots(1,5)(1,6)(1,7)(1,8)(2,5)\ldots$$

Будет ли при однородном методе, основанном на этой последовательности, выполняться в среднем больше или меньше сравнений, чем при быстрой сортировке?

66. [*M29*] При быстрой сортировке в наихудшем случае выполняется $\binom{N}{2}$ сравнений. Верно ли, что все алгоритмы ограниченной однородной сортировки (в смысле упр. 63) выполняют $\binom{N}{2}$ сравнений в наихудшем случае?

67. [*M48*] (Х. Л. Бьюс (H. L. Beus).) Верно ли, что в алгоритме быстрой сортировки предполагается минимальное среднее число сравнений среди всех алгоритмов ограниченной однородной сортировки?

68. [*25*] Докторская диссертация Говарда Б. Демута (Howard B. Demuth) "Electronic Data Sorting" (Stanford University, October, 1956) была, вероятно, первой работой, в которой сколько-нибудь детально рассматривались вопросы сложности вычислений в приложениях этого класса. Демут рассмотрел несколько абстрактных моделей устройств для сортировки и нашел нижние и верхние оценки среднего и максимального времени выполнения, достижимого в каждой модели. Простейшая его модель — "циклическая нереверсивная память" (рис. 61) — станет темой этого упражнения.

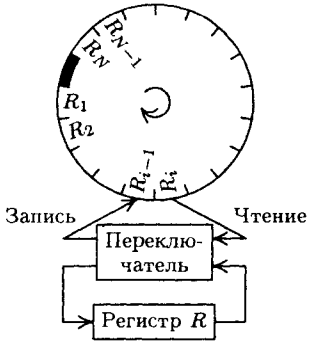


Рис. 61. Устройство, для которого стратегия метода пузырька является оптимальной.

Рассмотрим машину, которая сортирует $R_1 R_2 \ldots R_N$ за ряд проходов, где каждый проход состоит из следующих $N + 1$ шагов.

Шаг 1. Установить $R \leftarrow R_1$. (R — внутренний регистр машины.)

Шаг i для $1 \le i \le N$. Либо (i) установить $R_{i-1} \leftarrow R$, $R \leftarrow R_i$, либо (ii) установить $R_{i-1} \leftarrow R_i$, оставив R неизмененным.

Шаг $N+1$. Установить $R_N \leftarrow R$.

Задача состоит в том, чтобы найти такой метод выбора между альтернативами (i) и (ii), чтобы минимизировать число проходов, необходимых для сортировки.

Докажите, что метод пузырька оптимален для этой модели. Другими словами, покажите, что при стратегии, которая выбирает альтернативу (i), если $R \leq R_i$, и альтернативу (ii), если $R > R_i$, достигается минимальное число проходов.

И будут в смущении плетущие сети...

*— Книга пророка Исаии (19:9)**

* Здесь приведен перевод цитаты из "Ветхого Завета" на английском языке, использованный автором в оригинале настоящего издания. В каноническом издании "Библии" на русском языке этот стих звучит следующим образом: "И будут в смущении обрабатывающие лен и ткачи белых полотен". — *Прим. перев.*

5.4. ВНЕШНЯЯ СОРТИРОВКА

Пришло время заняться интересными задачами, возникающими в том случае, когда записей больше, чем может поместиться в быстродействующую оперативную память. Внешняя сортировка в корне отличается от внутренней (хотя в обоих случаях необходимо расположить записи данного файла в порядке неубывания), и объясняется это тем, что время доступа к массиву на внешних носителях жестко ограничено. Структура данных должна быть такой, чтобы сравнительно медленные периферийные запоминающие устройства (на магнитных лентах, дисках, барабанах и т. д.) могли удовлетворить потребности алгоритма сортировки. Поэтому большинство изученных до сих пор методов внутренней сортировки (вставка, обмен, выбор) фактически бесполезно для внешней сортировки; необходимо рассмотреть всю проблему заново.

Предположим, например, что предназначенный для сортировки файл состоит из 5 000 записей $R_1\, R_2 \ldots R_{5000000}$ длиной по 20 слов (хотя ключи R_i не обязательно должны иметь такую длину). Как быть, если во внутренней памяти данной машины помещается одновременно только 1 000 из этих записей?

Сразу напрашивается такое решение: рассортировать каждый из пяти подфайлов $R_1 \ldots R_{1000000}$, $R_{1000001} \ldots R_{2000000}$, $\ldots$, $R_{4000001} \ldots R_{5000000}$ по отдельности и затем слить полученные подфайлы. К счастью, слияние оперирует только очень простыми структурами данных, а именно — линейными списками, обрабатывать которые можно последовательно, как стеки или очереди. Поэтому для реализации метода слияния годятся самые непритязательные устройства внешней памяти.

Только что описанный процесс — внутренняя сортировка с последующим "внешним слиянием" — весьма популярен, и наше изучение внешней сортировки сведется, в основном, к вариациям на эту тему.

Возрастающие последовательности записей, получаемые на начальной фазе внутренней сортировки, в литературе о сортировке часто называются *цепочками* (string). Этот термин довольно широко распространен, но, к сожалению, он противоречит еще более популярному термину "string" в других разделах вычислительной науки, в которых он означает *произвольную* последовательность символов. При изучении перестановок уже было дано вполне подходящее название для упорядоченных сегментов файла, которые мы договорились называть восходящими сериями или просто *сериями*. В соответствии с этим будем использовать слово "серии" для обозначения упорядоченных частей файла. Таким образом, использование понятий "цепочки серий" и "серии цепочек" не приведет ни к каким недоразумениям.

Рассмотрим сначала процесс внешней сортировки, использующей в качестве вспомогательной памяти *накопители на магнитных лентах* (в дальнейшем для краткости будем употреблять термин "магнитные ленты"). Вероятно, простейшим и наиболее привлекательным способом слияния с помощью лент служит *сбалансированное двухпутевое слияние*, в основе которого лежит идея, использовавшаяся ранее в алгоритмах 5.2.4, N, S и L. В процессе слияния нам потребуются четыре "рабочие ленты". На первой фазе восходящие серии, получаемые при внутренней сортировке, помещаются поочередно на ленты 1 и 2 до тех пор, пока не исчерпаются исходные данные. Затем ленты 1 и 2 перематываем к началу и выполняем слияние серий, которые находятся на этих лентах, получая новые серии, вдвое длиннее исходных. Эти новые серии записываются по мере их формирования попеременно на

ленты 3 и 4. (Если на ленте 1 на одну серию больше, чем на ленте 2, то предполагается, что лента 2 содержит дополнительную "фиктивную" серию длиной 0.) Затем все ленты перематываются к началу и содержимое лент 3 и 4 сливается в серии удвоенной длины, записываемые поочередно на ленты 1 и 2. Процесс продолжается (при этом длина серий каждый раз удваивается) до тех пор, пока не останется одна серия (а именно — весь упорядоченный файл). Если после внутренней сортировки было получено S серий, причем $2^{k-1} < S \leq 2^k$, то процедура сбалансированного двухпутевого слияния произведет ровно $k = \lceil \lg S \rceil$ проходов по всем данным.

Например, в рассмотренной выше ситуации, когда требуется упорядочить 5 млн записей, а объем внутренней памяти составляет 1 млн записей, мы имеем $S = 5$. На начальной распределительной фазе процесса сортировки пять серий будут помещены на ленты следующим образом:

$$\begin{array}{ll} \text{Лента 1} & R_1 \ldots R_{1000000};\ R_{2000001} \ldots R_{3000000};\ R_{4000001} \ldots R_{5000000} \\ \text{Лента 2} & R_{1000001} \ldots R_{2000000};\ R_{3000001} \ldots R_{4000000} \\ \text{Лента 3} & \text{(Пустая)} \\ \text{Лента 4} & \text{(Пустая)} \end{array} \tag{1}$$

После первого прохода слияния на лентах 3 и 4 получатся более длинные серии, чем на лентах 1 и 2:

$$\begin{array}{ll} \text{Лента 3} & R_1 \ldots R_{2000000};\ R_{4000001} \ldots R_{5000000} \\ \text{Лента 4} & R_{2000001} \ldots R_{4000000} \end{array} \tag{2}$$

(В конец ленты 2 неявно добавляется фиктивная серия, так что серия $R_{4000001} \ldots R_{5000000}$ просто копируется на ленту 3.) После перемотки всех лент к началу следующий проход по данным приведет к такому результату:

$$\begin{array}{ll} \text{Лента 1} & R_1 \ldots R_{4000000} \\ \text{Лента 2} & R_{4000001} \ldots R_{5000000} \end{array} \tag{3}$$

(Серия $R_{4000001} \ldots R_{5000000}$ снова копируется, но если бы мы начали с 8 млн записей, в этот момент на ленте 2 содержалось бы $R_{4000001} \ldots R_{8000000}$.) Например, после еще одной перемотки на ленте 3 окажется серия $R_1 \ldots R_{5000000}$ и сортировка закончится.

Сбалансированное слияние легко обобщается для T лент при любом $T \geq 3$. Выберем произвольное число P, $1 \leq P < T$, и разделим T лент на два "банка": P лент в левом банке и $T - P$ лент в правом банке. Распределим исходные серии как можно равномернее по P лентам левого "банка", затем выполним P-путевое слияние слева направо, после этого — $(T - P)$-путевое слияние справа налево и так до тех пор, пока сортировка не завершится. Обычно значение P лучше всего выбирать равным $\lceil T/2 \rceil$ (см. упр. 3 и 4).

При $T = 4$, $P = 2$ имеем частный случай — сбалансированное двухпутевое слияние. Вновь рассмотрим предыдущий пример, используя большее количество лент; положим $T = 6$ и $P = 3$. Начальное распределение теперь будет таким:

$$\begin{array}{ll} \text{Лента 1} & R_1 \ldots R_{1000000};\ R_{3000001} \ldots R_{4000000} \\ \text{Лента 2} & R_{1000001} \ldots R_{2000000};\ R_{4000001} \ldots R_{5000000} \\ \text{Лента 3} & R_{2000001} \ldots R_{3000000} \end{array} \tag{4}$$

Первый проход слияния приведет к следующему результату:

$$\begin{array}{ll} \text{Лента 4} & R_1 \ldots R_{3000000} \\ \text{Лента 5} & R_{3000001} \ldots R_{5000000} \\ \text{Лента 6} & \text{(Пустая)} \end{array} \tag{5}$$

(Предполагается, что на ленте 3 помещена фиктивная серия.) На втором проходе слияния работа завершается и серии $R_1 \ldots R_{5000000}$ помещаются на ленту 1. Этот частный случай для $T = 6$ эквивалентен случаю для $T = 5$, поскольку лента 6 используется лишь при $S \geq 7$.

При трехпутевом слиянии затрачивается фактически несколько больше времени центрального процессора, чем при двухпутевом, но оно обычно пренебрежимо мало по сравнению со временем, необходимым для чтения, записи и перемотки ленты. Мы довольно хорошо оценим время выполнения сортировки, если примем во внимание только суммарную величину перемещений лент. В предыдущем примере ((4) и (5)) требуются только два прохода по данным по сравнению с тремя проходами при $T = 4$. Таким образом, слияние при $T = 6$ займет около двух третей времени по отношению к предыдущему случаю.

Сбалансированное слияние кажется очень простым и естественным. Но если приглядеться внимательнее, то сразу видно, что это не *наилучший* способ для рассмотренных выше частных случаев. Вместо того чтобы переходить от (1) к (2) и перематывать все ленты, нам следовало остановить первое слияние, когда на лентах 3 и 4 содержались соответственно $R_1 \ldots R_{2000000}$ и $R_{2000001} \ldots R_{4000000}$, а лента 1 была готова к считыванию $R_{4000001} \ldots R_{5000000}$. Затем ленты 2–4 могли быть перемотаны к началу и сортировка завершилась бы трехпутевым слиянием на ленту 2. Общее число записей, прочитанных с ленты в ходе этой процедуры, составило бы $4000000 + 5000000 = 9000000$ против $5000000 + 5000000 + 5000000 = 15{,}000000$ в сбалансированной схеме. Сообразительная машина могла бы постичь и это!

Имея пять серий и четыре ленты, можно поступить еще лучше, распределив серии следующим образом:

$$\begin{array}{ll} \text{Лента 1} & R_1 \ldots R_{1000000};\ R_{3000001} \ldots R_{4000000} \\ \text{Лента 2} & R_{1000001} \ldots R_{2000000};\ R_{4000001} \ldots R_{5000000} \\ \text{Лента 3} & R_{2000001} \ldots R_{3000000} \\ \text{Лента 4} & \text{(Пустая)} \end{array}$$

Теперь, выполнив трехпутевое слияние на ленту 4 и перемотку лент 3 и 4 с последующим трехпутевым слиянием на ленту 3, можно было бы завершить сортировку, прочитав всего $3000000 + 5000000 = 8000000$ записей.

Наконец, если бы мы имели шесть лент, то могли бы, конечно, записать исходные серии на ленты 1–5 и закончить сортировку за один проход, выполнив пятипутевое слияние на ленту 6. Анализ этих случаев показывает, что простое сбалансированное слияние не является наилучшим и было бы интересно поискать более удачные способы слияния.

В последующих разделах этой главы внешняя сортировка исследуется более глубоко. В разделе 5.4.1 рассматривается фаза внутренней сортировки, порождающая начальные серии. Особый интерес представляет технология “выбор с замещением”,

в которой используется порядок, присутствующий в большинстве данных, чтобы породить длинные серии, значительно превосходящие емкость внутренней памяти. В разделе 5.4.1 обсуждаются также структуры данных, удобные для многопутевого слияния.

Важнейшие схемы слияния рассматриваются в разделах 5.4.2–5.4.5. Пока мы не вступим в единоборство с грубой действительностью работающих накопителей на магнитных лентах и реальных сортируемых данных, лучше, изучая характеристики этих схем, ограничиться весьма приближенным представлением о сортировке на лентах. Например, можно с легкой душой полагать (как мы делали до сих пор), что первоначальные исходные записи появляются волшебным образом в течение первой распределительной фазы. На самом деле они, вероятно, будут занимать одну из наших лент и, быть может, даже целиком заполнят несколько бобин, так как лента не бесконечна! Лучше всего пренебречь подобными техническими деталями до тех пор, пока не будет достигнуто "академическое" понимание классических схем слияния. Затем в разделе 5.4.6 мы "вернемся на землю", рассмотрев практические ограничения, которые существенно влияют на выбор схемы слияния. В разделе 5.4.6 сравниваются основные схемы слияния из разделов 5.4.2–5.4.5 с учетом множества разнообразных предположений, которые встречаются на практике.

Иные подходы к проблеме внешней сортировки, *не* основанные на слиянии, обсуждаются в разделах 5.4.7 и 5.4.8. Анализ разнообразных аспектов внешней сортировки заканчивается в разделе 5.4.9, в котором рассматривается важная проблема сортировки с использованием таких устройств внешней памяти, как магнитные диски и барабаны.

Когда шла работа над первым изданием этой книги, накопители на магнитной ленте использовались повсеместно, в то время как магнитные диски считались слишком уж дорогой экзотикой. Но с начала 80-х годов цена на устройства памяти с магнитными дисками разительно снизилась, и в конце 90-х годов они практически вытеснили накопители на магнитных лентах в подавляющем большинстве компьютерных систем. Таким образом, вопрос, который еще совсем недавно считался важнейшим в проблеме сортировки (а именно — разработка и анализ методов сортировки применительно к особенностям функционирования накопителей на магнитных лентах), теперь представляется не таким уж важным.

Однако большинство подобных схем сортировки настолько изящны, а соответствующие алгоритмы так отражают результаты самых глубоких исследований, выполненных в ранние годы компьютеризации, что делать их достоянием только истории науки представляется слишком большим расточительством. Поэтому мы довольно подробно проанализируем схемы слияния и это, возможно, будет их последним появлением на сцене перед тем, как занавес за ними опустится окончательно.

Из всего, что известно нам сегодня,
вполне можно сделать следующий вывод:
эти методы сохранят свою актуальность и в дальнейшем.

— ПАВЕЛ КЕРТИС (1997)

УПРАЖНЕНИЯ

1. [*15*] В тексте раздела внешняя сортировка рассматривается после внутренней. Почему нельзя вообще покончить с фазой внутренней сортировки, просто выполняя слияние записей во все более и более длинные серии с самого начала?

2. [*10*] Каким будет содержимое лент (аналогичное (1)–(3)), если записи $R_1\ R_2 \ldots R_{5000000}$ сортируются с помощью трехленточного сбалансированного метода при $P = 2$? Сравните этот случай со слиянием на четырех лентах; сколько проходов по всем данным будет сделано после первоначального распределения серий?

3. [*20*] Покажите, что метод сбалансированного $(P, T-P)$-путевого слияния, примененный к S начальных серий, приводит к $2k$ проходам, если $P^k(T-P)^{k-1} < S \le P^k(T-P)^k$, и к $2k+1$ проходам, если $P^k(T-P)^k < S \le P^{k+1}(T-P)^k$.

Дайте простые формулы для вычисления (a) точного числа проходов как функции от S при $T = 2P$, (b) приближенного числа проходов при $S \to \infty$ для любых P и T.

4. [*HM15*] При каком значении P, $1 \le P < T$, значение $P(T-P)$ максимально?

5.4.1. Многопутевое слияние и выбор с замещением

В разделе 5.2.4 рассматривались методы внутренней сортировки, основанные на двухпутевом слиянии — процессе объединения двух упорядоченных последовательностей в одну. Нетрудно расширить этот анализ и на P-путевое слияние, когда P входных серий объединяются в одну выходную.

Пусть имеется P возрастающих серий, т. е. последовательностей записей, ключи которых расположены в порядке неубывания. Очевидным способом их слияния будет следующий: просмотреть первые записи каждой серии и выбрать из них ту, которая имеет минимальный ключ; эта запись передается на выход и исключается из входных данных, затем процесс повторяется. В любой момент времени потребуется просмотреть только P ключей (один на каждую серию) и выбрать из них наименьший. Если наименьшими окажутся два или более ключей, выбрать можно любой из них.

Пока P не слишком велико, этот выбор удобно осуществлять, просто выполняя $P-1$ сравнений для нахождения наименьшего из текущих ключей. Но если P равно, скажем, 8, то можно ускорить работу, используя *дерево выбора*, как описано в разделе 5.2.3; затем каждый раз потребуется примерно $\lg P$ сравнений (после начального формирования дерева). Рассмотрим, например, четырехпутевое слияние с двухуровневым деревом выбора.

Шаг 1

$$087 \begin{cases} 087 \begin{cases} 087\ 503\ \ \infty \\ 170\ 908\ \ \infty \end{cases} \\ 154 \begin{cases} 154\ 426\ 653\ \ \infty \\ 612\ \ \infty \end{cases} \end{cases}$$

Шаг 2

$$087\ 154 \begin{cases} 170 \begin{cases} 503\ \ \infty \\ 170\ 908\ \ \infty \end{cases} \\ 154 \begin{cases} 154\ 426\ 653\ \ \infty \\ 612\ \ \infty \end{cases} \end{cases}$$

Шаг 3 087 154 170 $\begin{cases} 170 \begin{cases} 503 \ \infty \\ 170\ 908 \ \infty \end{cases} \\ 426 \begin{cases} 426\ 653 \ \infty \\ 612 \ \infty \end{cases} \end{cases}$

$\vdots$

Шаг 9 087 154 170 426 503 612 653 908 ∞ $\begin{cases} \infty \begin{cases} \infty \\ \infty \end{cases} \\ \infty \begin{cases} \infty \\ \infty \end{cases} \end{cases}$

В этом примере в конце каждой серии помещен добавочный ключ "∞", чтобы слияние заканчивалось естественно. Так как внешнее слияние обычно имеет дело с очень длинными сериями, эта добавочная запись с ключом "∞" не увеличит существенно длину данных или объем работы при слиянии. Кроме того, такая "концевая" запись часто служит удобным способом разделения записей файла.

В рассматриваемом процессе каждый шаг, кроме первого, заключается в замещении наименьшего элемента следующим элементом из этой же серии и в изменении соответствующего пути в дереве выбора. Так, на шаге 2 изменяются 3 узла, которые содержали 087 на шаге 1; на шаге 3 изменяются 3 узла, содержавшие 154 на шаге 2, и т. д. Процесс замещения в дереве выбора одного ключа другим называется ***выбором с замещением***.

Мы можем по-разному рассматривать описанный процесс четырехпутевого слияния. С одной точки зрения он эквивалентен трем двухпутевым слияниям, выполняемым совместно, как сопрограммы; каждый узел дерева выбора изображает одну из последовательностей, используемых в процессах слияния. Дерево выбора, по существу, используется как приоритетная очередь с дисциплиной "наименьший из включенных первым исключается". Так же, как в разделе 5.2.3, можно было бы для реализации приоритетной очереди использовать не дерево выбора, а пирамиду. (Пирамиду, конечно, следовало бы организовать таким образом, чтобы на ее вершине появился *наименьший* элемент, а не наибольший, обратив для этого порядок соотношения 5.2.3–(3).) Так как пирамида не имеет фиксированного размера, можно избежать использования ключа "∞"; слияние заканчивается, когда пирамида становится пустой. С другой стороны, в приложениях, в которых используется внешняя сортировка, обычно имеют дело со сравнительно длинными записями и ключами, так что в пирамиду будут записаны вместо самих ключей указатели на них. Мы увидим ниже, что деревья выбора можно настолько удобно изображать с помощью указателей, что они (деревья), вероятно, в данной ситуации лучше пирамид.

Дерево "проигравших". На рис. 62 изображено полное бинарное дерево с 12 внешними (квадратными) узлами и 11 внутренними (круглыми); во внешних узлах записаны ключи, во внутренних — "победители", если дерево рассматривать как турнир для выбора наименьшего ключа. Меньшие числа над каждым узлом показывают традиционный способ распределения последовательных позиций памяти для полного бинарного дерева.

Чтобы определить новое состояние дерева выбора, изображенного на рис. 62, когда наименьший ключ 061 будет заменен другим ключом, нужно рассмотреть только ключи 512, 087 и 154. Если интерпретировать дерево как турнир, последние три ключа будут представлять собой проигравших в матчах с игроком 061. Это

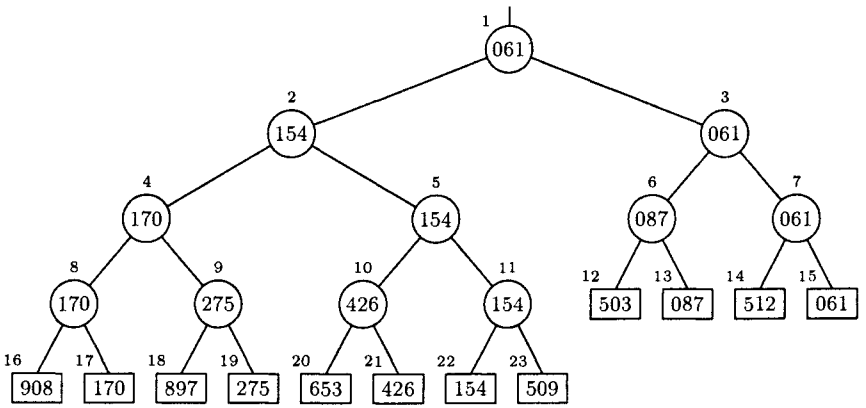


Рис. 62. Турнир, в котором выбирается наименьший ключ; используется полное бинарное дерево, узлы которого пронумерованы от 1 до 23.

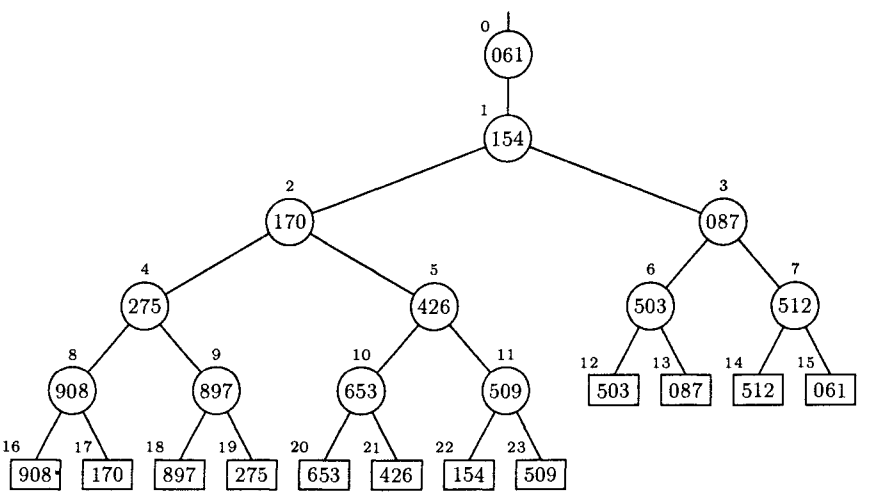


Рис. 63. Тот же турнир, что и на рис. 62, но показаны проигравшие, а не победители; чемпион находится на самом верху.

наводит на мысль, что мы в действительности должны записать во внутренние узлы *проигравшего* в каждом матче, а не победителя. Тогда информация, необходимая для изменения дерева, будет легкодоступной.

На рис. 63 изображено то же дерево, что и на рис. 62, но вместо победителей в нем представлены проигравшие. Дополнительный узел с номером "0" добавлен на вершине дерева для указания чемпиона турнира. Заметим, что каждый ключ, кроме ключа чемпиона, является проигравшим ровно один раз (см. раздел 5.3.3).

На практике внешним узлам в нижней части рис. 63 будут соответствовать весьма длинные записи, расположенные в памяти компьютера, а внутренним узлам — указатели на эти записи. Заметим, что P-путевое слияние требует ровно P

внешних и P внутренних узлов — по одному в соседних группах. Это наводит на мысль о возможности использовать известные эффективные методы распределения памяти. Нетрудно увидеть, каким образом можно применять дерево проигравших для выбора с замещением. Более детально мы обсудим этот алгоритм в настоящем разделе чуть позже.

Получение начальных серий посредством выбора с замещением. Технология выбора с замещением может использоваться на *первой* фазе внешней сортировки, если фактически выполнить P-путевое слияние входных данных с самими собой. В этом случае P выбирается достаточно большим, чтобы заполнить, по существу, всю внутреннюю память. Каждая запись при выводе замещается очередной записью из исходных данных. Если ключ новой записи меньше ключа выведенной записи, то мы не добавляем ее в текущую серию; в противном случае мы обычным образом включаем ее в дерево выбора, так что она образует часть серии, порождаемой в данный момент. Таким образом, каждая серия может содержать больше P записей, хотя в любой момент в дереве выбора находится не более P записей. В табл. 1 показан этот процесс для $P = 4$; числа, заключенные в скобки, ожидают включения в следующую серию.

Таблица 1

ПРИМЕР ЧЕТЫРЕХПУТЕВОГО ВЫБОРА С ЗАМЕЩЕНИЕМ

Содержимое памяти				Вывод
503	087	512	061	061
503	087	512	908	087
503	170	512	908	170
503	897	512	908	503
(275)	897	512	908	512
(275)	897	653	908	653
(275)	897	(426)	908	897
(275)	(154)	(426)	908	908
(275)	(154)	(426)	(509)	(Конец серии)
275	154	426	509	154
275	612	426	509	275

Этот важный метод формирования начальных серий впервые был описан Х. Г. Сьювордом [E. H. Seward, Master's Thesis, Digital Computer Laboratory Report R-232 (Mass. Inst. of Technology, 1954), 29–30]. Он привел соображения в пользу того, что если применять метод к случайным данным, серии, видимо, будут содержать более $1.5P$ записей. Ту же идею предложил примерно в 1950 году А. И. Думи (A. I. Dumey), который занимался разработкой в Engineering Research Associates специального устройства для сортировки, но не опубликовал ее. Само название "выбор с замещением" было придумано Э. Г. Френдом [E. H. Friend, *JACM* **3** (1956), 154], который заметил, что "ожидаемая длина порождаемой последовательности не поддается точной формулировке, но эксперименты позволяют предположить, что разумной оценкой будет $2P$".

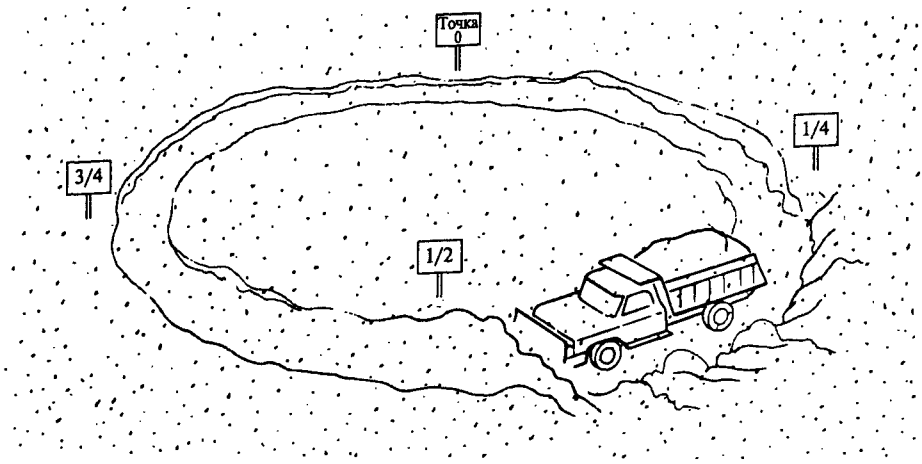


Рис. 64. Вечный снегоочиститель в своем нескончаемом цикле.

Э. Ф. Мур (E. F. Moore) предложил изящный способ доказательства того, что ожидаемая длина серии равна $2P$, проведя аналогию со снегоочистителем, движущимся по кругу [*U.S. Patent 2983904* (1961), columns 3–4]. Рассмотрим ситуацию, изображенную на рис. 64: на кольцевую дорогу равномерно падают снежинки и один снегоочиститель непрерывно убирает снег. Снег исчезает из системы, как только он выбрасывается на обочину. Точки дороги обозначаются вещественными числами x, $0 \le x < 1$, снежинка, падающая в точку x, соответствует входной записи с ключом x, а снегоочиститель представляет собой вывод процесса выбора с замещением. Скорость снегоочистителя обратно пропорциональна весу снега, который встречается на его пути, и ситуация вполне уравновешена, так что общее количество снежинок на дороге в точности равно P. Каждый раз, когда снегоочиститель проходит точку 0, на выходе формируется новая серия.

Интуитивно ясно, что система, поработав некоторое время, выйдет на установившийся режим, при котором снегоочиститель будет двигаться с постоянной скоростью (в силу круговой симметрии дороги). Это означает, что в точке, в которой находится снегоочиститель, снег имеет постоянный вес, а впереди снегоочистителя этот вес линейно уменьшается, как показано на рис. 65. Отсюда следует, что количество снега, удаляемого за один оборот (а именно — длина серии), вдвое превосходит количество снега P, присутствующего в любой момент.

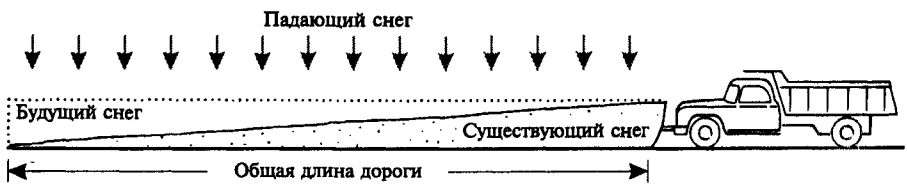


Рис. 65. Поперечное сечение, показывающее переменный вес снега перед снегоочистителем, когда система находится в установившемся режиме.

Во многих коммерческих приложениях входные данные *нельзя* считать полностью случайными — в них уже существует определенный порядок. Следовательно, серии, порождаемые выбором с замещением, преимущественно содержат больше чем 2P записей. В дальнейшем мы увидим, что время, необходимое для внешней сортировки методом слияния, в значительной степени зависит от количества серий, порождаемых начальной распределительной фазой, так что выбор с замещением становится особенно привлекательным. Другие способы внутренней сортировки порождали бы примерно вдвое больше начальных серий, поскольку размеры оперативной памяти ограничены.

Теперь детально рассмотрим процесс создания начальных серий методом выбора с замещением. Следующий алгоритм принадлежит Джону Р. Уолтерсу (John R. Walters), Джеймсу Пэйнтеру (James Painter) и Мартину Залку (Martin Zalk), которые использовали его в программе сортировки методом слияния для компьютера Philco 2000 в 1958 году. Алгоритм включает интересный способ начального формирования дерева выбора и разделения записей, принадлежащих разным сериям, а также вывода последней серии по единообразной, сравнительно простой логике. (Надлежащая обработка последней серии, порожденной методом выбора с замещением, оказывается довольно сложной; блок, осуществляющий эту обработку, часто бывает камнем преткновения для программиста.) Основная идея заключается в том, чтобы рассматривать каждый ключ как пару (S, K), где K — первоначальный ключ, а S — номер серии, которой принадлежит данная запись. Мы получим выходную последовательность, порожденную методом выбора с замещением, если лексикографически упорядочим эти расширенные ключи (S считается старше K).

В приведенном ниже алгоритме для представления дерева выбора используется структура данных, состоящая из P узлов. Предполагается, что j-й узел $X[j]$ содержит c слов, начинающихся с $\texttt{LOC}(X[j]) = L_0 + cj$, $0 \le j < P$. Он представляет как внутренний узел с номером j, так и внешний узел с номером $P + j$ (см. рис. 63). В каждом узле имеется несколько полей:

- `KEY` — ключ, находящийся в данном внешнем узле;
- `RECORD` — запись, находящаяся в данном внешнем узле (включая `KEY` как подполе);
- `LOSER` — указатель на "проигравшего" в данном внутреннем случае;
- `RN` — номер серии, содержащей запись, на которую указывает `LOSER`;
- `PE` — указатель на внутренний узел, расположенный в дереве выбора выше данного внешнего узла;
- `PI` — указатель на внутренний узел, расположенный в дереве выбора выше данного внутреннего узла.

Например, при $P = 12$ внутренний узел с номером 5 и внешний узел с номером 17 на рис. 63 будут представлены узлом $X[5]$ с полями $\texttt{KEY} = 170$, $\texttt{LOSER} = L_0 + 9c$ (адрес внешнего узла с номером 21), $\texttt{PE} = L_0 + 8c$, $\texttt{PI} = L_0 + 2c$.

Значения в полях `PE` и `PI` являются константами; таким образом, строго говоря, нет необходимости хранить их в явном виде. Однако иногда на начальной фазе внешней сортировки для быстрой работы с устройствами ввода-вывода выгоднее хранить эту избыточную информацию, чем вычислять ее каждый раз заново.

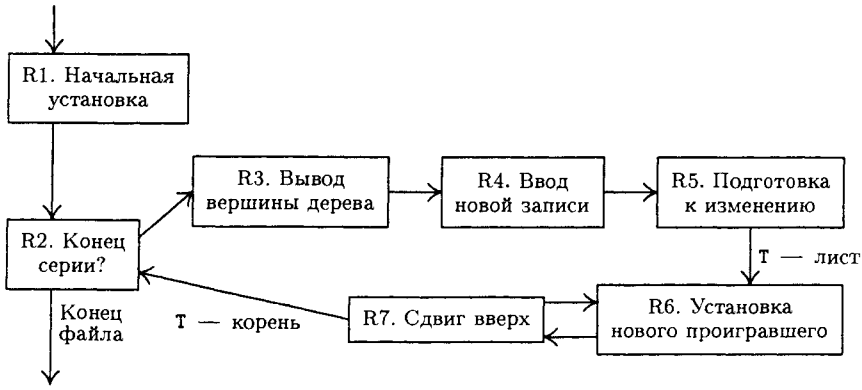


Рис. 66. Получение начальных серий методом выбора с замещением.

Алгоритм R (*Выбор с замещением*). Этот алгоритм (рис. 66) считывает записи последовательно из входного файла и записывает их последовательно в выходной файл, производя RMAX серий длиной P или больше (за исключением последней серии). Имеется $P \geq 2$ узлов $X[0], \ldots, X[P-1]$ с полями, описанными выше.

R1. [Начальная установка.] Установить RMAX $\leftarrow 0$, RC $\leftarrow 0$, LASTKEY $\leftarrow \infty$, Q $\leftarrow$ LOC($X[0]$) и RQ $\leftarrow 0$. (RC — номер текущей серии, а LASTKEY — ключ последней выведенной записи. Начальное значение LASTKEY должно быть больше любого возможного ключа; см. упр. 8.) Для $0 \leq j < P$, если обозначить J = LOC($X[j]$), начальное содержимое $X[j]$ установить следующим образом:

$$\texttt{LOSER(J)} \leftarrow \texttt{J}; \qquad \texttt{RN(J)} \leftarrow 0;$$
$$\texttt{PE(J)} \leftarrow \texttt{LOC}(X[\lfloor (P+j)/2 \rfloor]); \qquad \texttt{PI(J)} \leftarrow \texttt{LOC}(X[\lfloor j/2 \rfloor]).$$

(Установка LOSER(J) и RN(J) представляет собой искусственный способ образования начального дерева путем рассмотрения фиктивной серии с номером 0, которая никогда не выводится. Это — своего рода трюк; см. упр. 10.)

R2. [Конец серии?] Если RQ = RC, то перейти к шагу R3. (В противном случае RQ = RC + 1 и обработка серии с номером RC завершена. Здесь следует выполнить те специальные действия, которые соответствуют схеме слияния для последующих этапов сортировки.) Если RQ > RMAX, то выполнение алгоритма завершено; в противном случае установить RC $\leftarrow$ RQ.

R3. [Вывод вершины дерева.] (Сейчас Q указывает на "чемпиона" и RQ — номер его серии.) Если RQ $\neq 0$, то вывести RECORD(Q) и установить LASTKEY $\leftarrow$ KEY(Q).

R4. [Ввод новой записи.] Если входной файл исчерпан, установить RQ $\leftarrow$ RMAX + 1 и перейти к шагу R5. В противном случае поместить новую запись из входного файла в RECORD(Q). Если KEY(Q) < LASTKEY (т. е. эта запись не принадлежит текущей серии), то RQ $\leftarrow$ RQ+1, и теперь, если RQ > RMAX, установить RMAX $\leftarrow$ RQ.

R5. [Подготовка к изменению.] (Сейчас Q указывает на новую запись с номером серии RQ.) Установить T $\leftarrow$ PE(Q). (T — переменный указатель, который будет двигаться по дереву.)

R6. [Установка нового проигравшего.] Если RN(T) < RQ или если RN(T) = RQ и KEY(LOSER(T)) < KEY(Q), то поменять местами LOSER(T) ↔ Q, RN(T) ↔ RQ. (В переменных Q и RQ запоминается текущий победитель и номер его серии.)

R7. [Сдвиг вверх.] Если T = LOC(X[1]), то вернуться к шагу R2; в противном случае T ← PI(T) и следует вернуться к R6. ▌

В алгоритме R говорится о вводе и выводе записей по одной, тогда как практически лучше считывать и записывать относительно большие блоки записей. Следовательно, на самом деле за кулисами прячутся буфера ввода и вывода; их присутствие в памяти приводит к уменьшению значения P. Это будет пояснено в разделе 5.4.6.

***Преобразование серий с задержкой.** В работе R. J. Dinsmore, *CACM* **8** (1965), 48, предложено интересное усовершенствование метода выбора с замещением, использующее понятие, которое будем называть *степенью свободы*. Как мы видели, в каждом блоке записей, находящемся на ленте в составе серии, содержатся записи в порядке неубывания, так что первый элемент — наименьший, а последний — наибольший. В обычном процессе выбора с замещением наименьший элемент каждого блока в некоторой серии всегда не меньше наибольшего элемента в предыдущем блоке данной серии. Это соответствует "первой степени свободы". Динсмор предложил ослабить такое условие до "m степеней свободы"; новое условие не требует, чтобы наименьший элемент каждого блока был не меньше, чем наибольший элемент предыдущего блока, но он *не должен быть меньше наибольших элементов каких-либо m предыдущих блоков той же серии*. Записи в отдельном блоке упорядочены, как и ранее, но соседние блоки не обязаны быть взаимно упорядоченными.

Предположим, например, что в блоках содержится только по две записи. Приведенная ниже последовательность блоков является серией с тремя степенями свободы:

$$| \ 08\ 50 \ | \ 06\ 90 \ | \ 17\ 27 \ | \ 42\ 67 \ | \ 51\ 89 \ | \qquad (1)$$

Следующий блок, который может быть частью этой серии, должен начинаться с элемента, не меньшего, чем третий по порядку элемент множества $\{50, 90, 27, 67, 89\}$, считая от наибольшего, т. е. не меньше 67. Последовательность (1) не является серией с двумя степенями свободы, так как 17 меньше, чем 50 и 90.

Серия с m степенями свободы в процессе чтения на следующей фазе сортировки может быть преобразована таким образом, что для всех практических целей она будет серией в обычном смысле. Начнем с чтения первых m блоков в m буферов и будем выполнять их m-путевое слияние; когда один из буферов исчерпается, поместим в него $(m+1)$-й блок, и т. д. Таким образом можно восстановить серию в виде одной упорядоченной последовательности, поскольку первое слово каждого вновь считываемого блока должно быть больше последнего слова только что исчерпанного блока или равно ему (если оно не было меньше, чем наибольшие элементы каких-либо предшествующих ему m блоков). Этот метод преобразования серии, в сущности, является m-путевым слиянием, использующим только одно устройство внешней памяти для всех входных блоков! Процедура преобразования действует, как сопрограмма, к которой обращаются каждый раз, когда нужно получить одну очередную запись серии. Можно в одно и то же время преобразовывать различные серии с различных устройств и с различными степенями свободы и сливать получающиеся серии. Это, по существу, подобно тому, как если бы процесс четырехпутевого

слияния, рассмотренный в начале настоящего раздела, был представлен в виде нескольких процессов двухпутевого слияния, которые происходят одновременно.

Такую остроумную идею трудно до конца проанализировать, но в работе T. O. Espelid, *BIT* **16** (1976), 133–142, показано, как распространить нашу аналогию с работой снегоочистителя на этот случай и получить соответствующую приближенную формулу. Как следует из выведенной формулы аппроксимации, длина серии будет равна примерно

$$2P + (m - 1.5)\left(\frac{2P + (m-2)b}{2P + (2m-3)b}\right) b,$$

если b — размер блока и $m \geq 2$. Эти данные довольно хорошо согласуются с проведенными практическими экспериментами. Такое увеличение длины нельзя считать достаточным для компенсации повышения сложности процесса. Но, с другой стороны, это может дать некоторые преимущества, когда существует возможность организовать довольно большое число буферов на второй фазе сортировки.

***Натуральный выбор.** Другой путь увеличения длины серий, порождаемых выбором с замещением, был исследован в работе W. D. Frazer, C. K. Wong, *CACM* **15** (1972), 910–913. Изложенная авторами идея состоит в том, чтобы следовать алгоритму R, кроме случая, когда в дерево включается новая запись, ключ которой меньше, чем LASTKEY. Тогда вывод направляется во внешний *дополнительный буфер* и считывается новая запись. Этот процесс продолжается до тех пор, пока в дополнительном буфере не окажется определенного количества записей P'. Тогда остаток текущей серии выводится из дерева, а элементы из дополнительного буфера используются в качестве исходных данных для следующей серии.

Рассмотренный метод должен порождать более длинные серии, чем метод выбора с замещением, поскольку он "обходит" вновь поступающие "мертвые" записи, вместо того чтобы позволять им заполнять дерево; но ему требуется дополнительное время на обмен с дополнительным буфером. Когда $P' > P$, некоторые записи могут оказаться в этом буфере дважды, но при $P' \leq P$ такого случиться не может.

Фрэйзер и Уон, проведя обширные эмпирические испытания своего метода, заметили, что P достаточно велико (скажем, $P \geq 32$) и $P' = P$, средняя длина серии для случайных данных оказывается равной eP, где $e \approx 2.718$ — основание натуральных логарифмов. Это явление, а также тот факт, что метод был получен в результате эволюции метода простого выбора с замещением, послужили авторам основанием для того, чтобы назвать свой метод *натуральным выбором*.

Можно доказать "натуральный" закон для длины серии, вновь воспользовавшись аналогией со снегоочистителем (см. рис. 64) и применив элементарный математический анализ. Пусть L обозначает длину пути, а $x(t)$ — положение снегоочистителя в момент t при $0 \leq t \leq T$. Предположим, что в момент T внешний буфер (резервуар) заполняется. В этот момент падение снега временно прекращается, пока снегоочиститель возвращается в исходное положение (счищая P снежинок, оставшихся на его пути). Ситуация такая же, как и ранее, только "условия равновесия" другие: вместо P снежинок на всей дороге мы имеем P снежинок перед снегоочистителем и резервуар (за снегоочистителем), заполняющийся до уровня, равного $P' = P$ снежинок. В течение времени dt снегоочиститель продвигается на dx, если выводятся $h(x,t)\,dx$ записей, где $h(x,t)$ — толщина слоя снега в момент времени t в

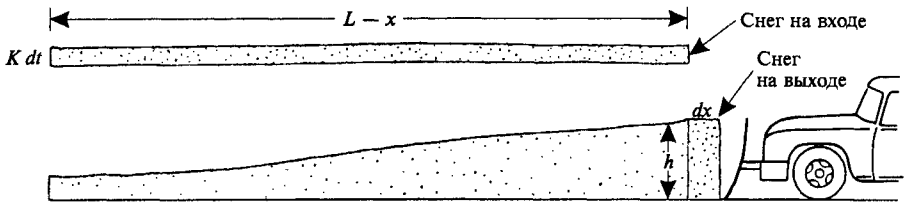


Рис. 67. Вводится и выводится равное количество снега; за время dt снегоочиститель перемещается на dx.

точке $x = x(t)$, измеряемая в соответствующих единицах. Следовательно, $h(x,t) = h(x,0) + Kt$ для всех x. Так как число записей в памяти остается постоянным, то $h(x,t)\,dx$ есть также число записей, вводимых *перед* снегоочистителем, а именно — $K\,dt\,(L-x)$, где K — скорость падения снега (рис. 67). Таким образом,

$$\frac{dx}{dt} = \frac{K(L-x)}{h(x,t)}. \tag{2}$$

К счастью, оказывается, что $h(x,t)$ — константа, равная KT при всех $x = x(t)$ и $0 \le t \le T$, так как снег падает равномерно в точку $x(t)$ в течение $T-t$ единиц времени после того, как снегоочиститель проходит эту точку, плюс t единиц времени перед тем, как он вернется. Иными словами, снегоочиститель все время видит перед собой одинаковый слой снега на протяжении всего пути, если допустить, что достигнут установившийся режим, т. е. этот путь всегда один и тот же. Следовательно, общее количество счищаемого снега (длина серии) составляет LKT, а количество снега в памяти есть количество снега, счищаемого после момента T, а именно — $KT\big(L - x(T)\big)$. Решением уравнения (2) при условии, что $x(0) = 0$, будет

$$x(t) = L(1 - e^{-t/T}). \tag{3}$$

Следовательно, $P = LKTe^{-1} = (\text{длина серии})/e$ — как раз то, что и требовалось доказать.

В упр. 21–23 показано, что данный анализ можно распространить на произвольное P'; например, когда $P' = 2P$, средняя длина серии оказывается равной $e^{\theta}(e - \theta)P$, где $\theta = \big(e - \sqrt{e^2-4}\big)/2$. Это результат, который вряд ли можно было предвидеть заранее! В табл. 2 приводится зависимость между длиной серии и размером дополнительного буфера. С помощью этой таблицы можно оценить эффективность натурального выбора для конкретного компьютера в той или иной ситуации. Строки таблицы, соответствующие размеру дополнительного буфера $< P$, получены с помощью несколько усовершенствованной методики, которая рассматривается в упр. 27.

Идеи преобразования серий с задержкой и натурального выбора можно скомбинировать, как описано в работе T. C. Ting, Y. W. Wang, *Comp. J.* **20** (1977), 298–301.

***Анализ выбора с замещением.** Вернемся теперь к выбору с замещением без вспомогательного буфера. Аналогия со снегоочистителем дает довольно хорошую оценку средней длины серий, получаемых при выборе с замещением "в пределе". Тем не менее можно получить значительно более точную информацию об алгоритме R, используя результаты анализа серий в перестановках, который выполнен в разде-

Таблица 2

ДЛИНА СЕРИЙ ПРИ НАТУРАЛЬНОМ ВЫБОРЕ

Размер дополнительного буфера	Длина серии	$k+\theta$	Размер дополнительного буфера	Длина серии	$k+\theta$
$0.10000P$	$2.15780P$	0.32071	$0.00000P$	$2.00000P$	0.00000
$0.50000P$	$2.54658P$	0.69952	$0.43428P$	$2.50000P$	0.65348
$1.00000P$	$2.71828P$	1.00000	$1.30432P$	$3.00000P$	1.15881
$2.00000P$	$3.53487P$	1.43867	$1.95014P$	$3.50000P$	1.42106
$3.00000P$	$4.16220P$	1.74773	$2.72294P$	$4.00000P$	1.66862
$4.00000P$	$4.69446P$	2.01212	$4.63853P$	$5.00000P$	2.16714
$5.00000P$	$5.16369P$	2.24938	$21.72222P$	$10.00000P$	4.66667
$10.00000P$	$7.00877P$	3.17122	$5.29143P$	$5.29143P$	2.31329

Величина $k+\theta$ определена в упр. 22 или (если $k=0$) в упр. 27.

ле 5.1.3. Для удобства будем считать, что входной файл является последовательностью произвольной длины независимых случайных чисел в диапазоне от 0 до 1.

Пусть

$$g_P(z_1, z_2, \ldots, z_k) = \sum_{l_1, l_2, \ldots, l_k \geq 0} a_P(l_1, l_2, \ldots, l_k) z_1^{l_1} z_2^{l_2} \ldots z_k^{l_k}$$

является производящей функцией для длины серии, которая получена при P-путевом выборе с замещением, примененном к такому файлу, где $a_P(l_1, l_2, \ldots, l_k)$ есть вероятность того, что первая серия имеет длину l_1, вторая — l_2, ..., k-я — l_k. Будем основываться на следующей "теореме независимости", так как она сводит наш анализ к случаю $P=1$.

Теорема K. $g_P(z_1, z_2, \ldots, z_k) = g_1(z_1, z_2, \ldots, z_k)^P$.

Доказательство. Пусть исходные ключи суть $X_1, X_2, X_3, \ldots$. Алгоритм R разделяет их на P подпоследовательностей в соответствии с тем, в какой внешний узел дерева они попадают. Подпоследовательность, содержащая X_n, определяется значениями $X_1, \ldots, X_{n-1}$. Таким образом, эти подпоследовательности являются независимыми последовательностями независимых случайных чисел, расположенных между 0 и 1. Кроме того, результат выбора с замещением в точности совпадает с результатом P-путевого слияния, если его выполнить над этими подпоследовательностями. Некоторый элемент принадлежит j-й серии подпоследовательности тогда и только тогда, когда он принадлежит j-й серии, полученной при выборе с замещением (так как на шаге R4 ключи `LASTKEY` и `KEY(Q)` принадлежат одной подпоследовательности).

Иначе говоря, можно считать, что алгоритм R применяется к P случайным независимым исходным файлам и что на шаге R4 считывается следующая запись из файла, соответствующего внешнему узлу `Q`. В этом смысле рассматриваемый алгоритм эквивалентен P-путевому слиянию, при котором концы серий отмечаются "убыванием" элементов.

Таким образом, на выходе будут получены серии длиной $(l_1, \ldots, l_k)$ тогда и только тогда, когда подпоследовательности состоят из серий длиной $(l_{11}, \ldots, l_{1k})$,

$\ldots, (l_{P1}, \ldots, l_{Pk})$, где l_{ij} — некоторые неотрицательные целые числа, удовлетворяющие соотношению $\sum_{1 \le i \le P} l_{ij} = l_j$ при $1 \le j \le k$. Отсюда получаем

$$a_P(l_1, \ldots, l_k) = \sum_{\substack{l_{11}+\cdots+l_{P1}=l_1 \\ \vdots \\ l_{1k}+\cdots+l_{Pk}=l_k}} a_1(l_{11}, \ldots, l_{1k}) \ldots a_1(l_{P1}, \ldots, l_{Pk}),$$

что эквивалентно искомому результату. ▌

В разделе 5.1.3 было проанализировано среднее значение L_k — длины k-й серии при $P = 1$ (эти значения приведены в табл. 5.1.3–2). Из теоремы K следует, что средняя длина k-й серии при любом P в P раз больше средней длины при $P = 1$; она равна $L_k P$. Дисперсия также в P раз больше, так что стандартное отклонение длины серии пропорционально $\sqrt{P}$. Эти результаты были впервые получены Б. Дж. Гэсснер (B. Y. Gassner) около 1958 года.

Таким образом, первая серия, полученная для случайных данных алгоритмом R, будет иметь длину, приближенно равную $(e-1)P \approx 1.718P$ записей, вторая — $(e^2 - 2e)P \approx 1.952P$, третья — $1.996P$; длина следующих серий будет очень близка к $2P$, пока мы не дойдем до последних двух серий (упр. 14). Стандартное отклонение длины большинства этих серий приближенно равно $\sqrt{(4e-10)P} \approx 0.934\sqrt{P}$ [*CACM* **6** (1963), 685–687]. Кроме того, согласно упр. 5.1.3–10 *суммарная* длина первых k серий будет довольно близка к $\left(2k - \frac{1}{3}\right)P$ со стандартным отклонением $\left(\left(\frac{2}{3}k + \frac{2}{9}\right)P\right)^{1/2}$. Производящие функции $g_1(z, z, \ldots, z)$ и $g_1(1, \ldots, 1, z)$ выводятся в упр. 5.1.3–9 и 11.

В приведенном выше анализе предполагалось, что исходный файл бесконечно длинный, но доказательство теоремы K показывает, что такая же вероятность $a_p(l_1, \ldots, l_k)$ получилась бы при наличии любой случайной исходной последовательности, содержащей, по крайней мере, $l_1 + \cdots + l_k + P$ элементов. Значит, полученные результаты применимы для файла размером, скажем, $N > (2k+1)P$ в силу малой величины стандартного отклонения.

В дальнейшем мы познакомимся с рядом приложений, в которых схема слияния требует, чтобы одни серии были восходящими, а другие — нисходящими. Поскольку остаток, накапливающийся в памяти у конца восходящей серии, как правило, содержит числа, в среднем меньшие, чем случайные данные, то изменение направления упорядочения приводит к уменьшению средней длины серий. Рассмотрим, например, снегоочиститель, который должен разворачиваться каждый раз по достижении конца прямой дороги; он будет очень быстро передвигаться по только что очищенному участку. В случае изменяемого направления длина серий для случайных данных изменяется между $1.5P$ и $2P$ (см. упр. 24).

УПРАЖНЕНИЯ

1. [*10*] Каким будет шаг 4 в примере для четырехпутевого слияния, приведенном в начале этого раздела?

2. [*12*] Какие изменения произошли бы в дереве на рис. 63, если бы ключ 061 был заменен ключом 612?

3. [*16*] (Э. Ф. Мур (E. F. Moore).) Что получится в результате применения четырехпутевого выбора с замещением к последовательным словам приведенного ниже предложения:

```
fourscore and seven years ago our fathers brought forth
on this continent a new nation conceived in liberty and
dedicated to the proposition that all men are created equal.
```

(Используйте обычный алфавитный порядок, рассматривая каждое слово как один ключ.)

4. [*16*] Примените четырехпутевой *натуральный* выбор к предложению из упр. 3, используя дополнительный буфер емкостью 4.

5. [*00*] Верно ли, что выбор с замещением, использующий дерево, работает, только если P есть степень 2 или сумма двух степеней 2?

6. [*15*] В алгоритме R указывается, что P должно быть ≥ 2. Какие относительно небольшие изменения необходимо сделать в этом алгоритме, чтобы он правильно работал для всех $P \geq 1$?

7. [*17*] Что делает алгоритм R в случае отсутствия исходной информации?

8. [*20*] Алгоритм R использует искусственный ключ "∞", который должен быть больше любого возможного ключа. Покажите, что если бы какой-нибудь реальный ключ оказался равным ∞, то алгоритм мог бы ошибиться, и объясните, как изменить алгоритм в случае, когда реализация "настоящего" ∞ неудобна.

▶ **9.** [*23*] Как бы вы изменили алгоритм R, чтобы он выводил одни заданные серии (в зависимости от RC) в порядке возрастания, а другие — в порядке убывания?

10. [*26*] Начальная установка указателей LOSER на шаге R1 обычно не соответствует никакому действительному турниру, так как внешний узел $P + j$ может не лежать в поддереве с вершиной во внутреннем узле j. Объясните, почему алгоритм R все равно работает. [*Указание.* Будет ли работать алгоритм R, если множеству {LOSER(LOC($X[0]$)), ..., LOSER(LOC($X[P-1]$))} присваивается на шаге R1 *произвольная* перестановка множества {LOC($X[0]$), ..., LOC($X[P-1]$)}?]

11. [*M25*] Верно ли утверждение, что для случайных исходных данных вероятность того, что KEY(Q) < LASTKEY на шаге R4, приближенно равна $\frac{1}{2}$?

12. [*M46*] Детально проанализируйте количество выполнений каждой части алгоритма R. Например, как часто выполняется перестановка на шаге R6?

13. [*13*] Почему вторая серия, полученная при выборе с замещением, обычно длиннее первой?

▶ **14.** [*HM25*] Используйте аналогию со снегоочистителем, чтобы оценить среднюю длину *двух последних серий*, которые получены при выборе с замещением, примененном к длинной последовательности исходных данных.

15. [*20*] Верно ли, что последняя серия, полученная при выборе с замещением, никогда не содержит более P записей? Проанализируйте свой ответ.

16. [*M26*] Найдите "простое" необходимое и достаточное условие того, что файл $R_1 R_2 \ldots R_N$ будет полностью упорядочен за один проход P-путевого выбора с замещением. Какова вероятность этого события как функции P и N, если исходными данными служит случайная перестановка множества $\{1, 2, \ldots, N\}$?

17. [*20*] Что получается в результате работы алгоритма R, когда исходные ключи представляют собой невозрастающую последовательность $K_1 \geq K_2 \geq \cdots \geq K_N$?

▶ **18.** [*22*] Что произойдет, если *вновь* применить алгоритм R к файлу, полученному в результате работы алгоритма R?

19. [*HM22*] Используйте аналогию со снегоочистителем, чтобы доказать, что первая серия, полученная при выборе с замещением, имеет длину примерно $(e-1)P$ записей.

20. [*HM24*] Какую примерно длину имеет первая серия, полученная при натуральном выборе, когда $P = P'$?

▶ **21.** [*HM23*] Определите приблизительную длину серий, полученных посредством натурального выбора при $P' < P$.

22. [*HM40*] Целью этого упражнения является определение средней длины серий, получаемых при натуральном выборе при $P' > P$. Пусть $\kappa = k + \theta$ — действительное число ≥ 1, где $k = \lfloor \kappa \rfloor$, а $\theta = \kappa \bmod 1$. Рассмотрим функцию $F(\kappa) = F_k(\theta)$, где $F_k(\theta)$ — полиномы, определяемые производящей функцией

$$\sum_{k \geq 0} F_k(\theta) z^k = e^{-\theta z}/(1 - z e^{1-z}).$$

Таким образом, $F_0(\theta) = 1$, $F_1(\theta) = e - \theta$, $F_2(\theta) = e^2 - e - e\theta + \frac{1}{2}\theta^2$ и т. д.

Предположим, что в момент времени 0 снегоочиститель начинает моделировать процесс натурального выбора, и допустим, что за T единиц времени позади него упадут ровно P снежинок. В этот момент второй снегоочиститель начинает тот же путь, занимая в момент времени $t + T$ то же положение, которое занимал первый снегоочиститель в момент t. В конце концов, к моменту κT позади первого снегоочистителя упадут ровно P' снежинок; второй снегоочиститель очищает остаток дороги и исчезает.

Используя эту модель для интерпретации натурального выбора, покажите, что длина серии $e^\theta F(\kappa) P$ получается при

$$P'/P = k + 1 + e^\theta \left(\kappa F(\kappa) - \sum_{j=0}^{k} F(\kappa - j) \right).$$

23. [*HM35*] В предыдущем упражнении проанализирован натуральный выбор в том случае, когда записи из резервуара всегда читаются в том же порядке, в котором они записывались: "первым включается — первым исключается". Оцените длину серий, которая получилась бы, если бы содержимое резервуара, оставшееся от предыдущей серии, читалось в совершенно *случайном* порядке, как будто записи в резервуаре тщательно перемешивались между сериями.

24. [*HM39*] Цель этого упражнения — анализ последствий, вызванных случайным изменением направления упорядочения серий в выборе с замещением.

a) Пусть $g_P(z_1, z_2, \ldots, z_k)$ — та же производящая функция, что и в теореме K, но для каждой из k серий определено, является ли она восходящей либо нисходящей. Например, мы можем считать, что все серии с нечетными номерами — восходящие, а с четными — нисходящие. Покажите, что теорема K справедлива для каждой из 2^k производящих функций такого вида.

b) В силу (a) можно считать $P = 1$. Можно также предположить, что исходной является равномерно распределенная последовательность независимых случайных величин в диапазоне между 0 и 1. Пусть

$$a(x, y) = \begin{cases} e^{1-x} - e^{y-x}, & \text{если } x \leq y; \\ e^{1-x}, & \text{если } x > y. \end{cases}$$

Пусть $f(x)\,dx$ — вероятность того, что определенная возрастающая серия начинается с x. Докажите, что $\left(\int_0^1 a(x, y) f(x)\,dx\right) dy$ есть вероятность того, что следующая серия начинается с y. [*Указание.* Рассмотрите для каждого $n \geq 0$ вероятность того, что $x \leq X_1 \leq \cdots \leq X_n > y$ при данных x и y.]

c) Рассмотрите серии, меняющие направление упорядочения с вероятностью p; другими словами, направление каждой серии, кроме первой, совпадает с направлением предыдущей серии с вероятностью $q = (1-p)$ и противоположно ему с вероятностью p. (Таким образом, если $p = 0$, то все серии имеют одинаковые направления; если $p = 1$, направление серий чередуется, а при $p = \frac{1}{2}$ серии случайны и независимы.) Пусть

$$f_1(x) = 1, \qquad f_{n+1}(y) = p\int_0^1 a(x,y)f_n(1-x)\,dx + q\int_0^1 a(x,y)f_n(x)\,dx.$$

Покажите, что вероятность того, что n-я серия начинается с x, есть $f_n(x)\,dx$, если $(n-1)$-я серия восходящая, и $f_n(1-x)\,dx$, если $(n-1)$-я серия нисходящая.

d) Найдите решение f для уравнения установившегося режима

$$f(y) = p\int_0^1 a(x,y)f(1-x)\,dx + q\int_0^1 a(x,y)f(x)\,dx, \qquad \int_0^1 f(x)\,dx = 1.$$

[*Указание.* Покажите, что $f''(x)$ не зависит от x.]

e) Покажите, что последовательность $f_n(x)$ п. (c) весьма быстро сходится к функции $f(x)$ п. (d).

f) Покажите, что средняя длина восходящей серии, начинающейся с x, равна e^{1-x}.

g) Наконец, объедините все предыдущие результаты и докажите следующую теорему. *Если направления последовательных серий при выборе с замещением независимо изменяются на противоположные с вероятностью p, то средняя длина серии стремится к $(6/(3+p))P$.*

(Эта теорема при $p = 1$ впервые была доказана Кнутом [*CACM* **6** (1963), 685–688]; при $p = \frac{1}{2}$ ее доказал А. Г. Конхейм (A. G. Konheim) в 1970 году.)

25. [*HM40*] Рассмотрите следующую процедуру.

N1. Прочитать запись, поместив ее в "резервуар" емкостью в одно слово. Затем прочитать следующую запись R, и пусть K будет ее ключом.

N2. Вывести содержимое резервуара, установить LASTKEY равным его ключу и очистить резервуар.

N3. Если K < LASTKEY, то вывести R, установить LASTKEY ← K и перейти к шагу N5.

N4. Если резервуар не пуст, вернуться к шагу N2; в противном случае поместить R в резервуар.

N5. Прочитать новую запись R и установить K равным ее ключу. Перейти к шагу N3. ∎

Эта процедура, в сущности, эквивалентна натуральному выбору с $P = 1$ и $P' = 1$ или 2 (в зависимости от того, в какой момент мы опустошаем резервуар — как только он заполнится или когда необходимо будет записать в заполненный резервуар новый элемент, переполняющий его), но она порождает *нисходящие* серии и ее выполнение никогда не прекращается. Такие отклонения не приносят вреда, они удобны для достижения нашей цели.

Действуя, как в упр. 24, обозначим через $f_n(x,y)\,dy\,dx$ вероятность того, что x и y суть значения LASTKEY и K соответственно сразу же после n-го выполнения шага N2. Докажите, что существует функция $g_n(x)$ от одной переменной, такая, что $f_n(x,y) = g_n(x)$, если $x < y$, и $f_n(x,y) = g_n(x) - e^{-y}(g_n(x) - g_n(y))$, если $x > y$. Функция $g_n(x)$ определяется соотношениями $g_1(x) = 1$:

$$g_{n+1}(x) = \int_0^x e^u g_n(u)\,du + \int_0^x dv\,(v+1)\int_v^1 du\,((e^v-1)g_n(u)+g_n(v))$$
$$+\,x\int_x^1 dv\int_v^1 du\,((e^v-1)g_n(u)+g_n(v)).$$

Покажите далее, что ожидаемая длина n-й серии равна

$$\int_0^1 dx\int_0^x dy\,(g_n(x)(e^y-1)+g_n(y))(2-\tfrac{1}{2}y^2)+\int_0^1 dx\,(1-x)g_n(x)e^x.$$

[*Замечание.* Решение этого уравнения в установившемся режиме оказывается очень сложным; оно было численно найдено Дж. Мак-Кенной (J. McKenna). Он показал, что длина серии стремится к предельной ≈ 2.61307209. Теорема K не применима к натуральному выбору, так что случай $P=1$ нельзя распространить на другие P.]

26. [*M33*] Рассмотрев алгоритм упр. 25 как определение натурального выбора для $P'=1$, найдите среднюю длину *первой* серии для $P'=r$ при любом $r\ge 0$ по следующей схеме.

a) Покажите, что первая серия имеет длину n с вероятностью

$$(n+r)\left[{n+r \atop n}\right]\Big/(n+r+1)!.$$

b) Определите "числа Стирлинга второго порядка" $[[{n \atop m}]]$ правилами

$$\left[\left[{0 \atop m}\right]\right]=\delta_{m0},\quad \left[\left[{n \atop m}\right]\right]=(n+m-1)\left(\left[\left[{n-1 \atop m}\right]\right]+\left[\left[{n-1 \atop m-1}\right]\right]\right)\qquad \text{при } n>0.$$

Докажите, что

$$\left[{n+r \atop n}\right]=\sum_{k=0}^{r}{n+r \choose k+r}\left[\left[{r \atop k}\right]\right].$$

c) Докажите, что средняя длина первой серии будет, следовательно, равна $c_r e - r - 1$, где

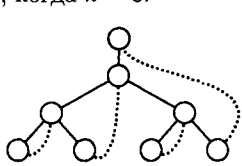
$$c_r=\sum_{k=0}^{r}\left[\left[{r \atop k}\right]\right]\frac{r+k+1}{(r+k)!}.$$

▶ **27.** [*HM30*] (В. Добосевич (W. Dobosiewicz).) Если при $P'<P$ используется натуральный выбор, то не нужно прекращать формирование серии после заполнения вспомогательного буфера. Можно сохранять записи, которые не принадлежат текущей серии, в главной приоритетной очереди, как при выборе с замещением, до тех пор, пока в текущей серии не останется только P' записей. Тогда их можно сбросить на выход и заменить содержимым вспомогательного буфера.

Насколько такой подход лучше более простого подхода, проанализированного в упр. 21?

28. [*25*] В основном тексте раздела рассматривается только сортировка записей фиксированного размера. Как приспособить выбор с замещением к записям *переменной длины*?

29. [*22*] Рассмотрим 2^k узлов полностью двоичного правопрошитого дерева, которое ниже показано графически в варианте, когда $k=3$:



(Ср. с 2.3.1–(10); верхний узел есть голова списка, пунктирными линиями показаны прошитые связи. В этом примере мы сосредоточим внимание на сортировке с использованием структуры полного двоичного дерева, когда узел 0 (аналог головы списка) добавляется перед узлом 1, как в "дереве проигравших" на рис. 63.)

Покажите, как организовать 2^{n+k} внутренних узлов большого дерева проигравших на этих 2^k основных узлах так, чтобы (i) каждый главный узел удерживал точно 2^n узлов большого дерева; (ii) в большом дереве подсоединенные узлы подключались либо к одному и тому же главному узлу, либо к главным узлам, подсоединенным рядом друг с другом; (iii) никакие две пары соседних узлов в большом дереве не разделялись одной и той же связью в главном дереве. [Таким образом, множество виртуальных процессоров в сети большого двоичного дерева можно отобразить на реальные процессоры без нежелательной избыточной перегрузки каналов связи.]

30. [*M29*] Докажите, что если $n \geq k \geq 1$, то построение в предыдущем упражнении оптимально в том смысле, что *любой* 2^k-узловой основной граф, удовлетворяющий условиям (i), (ii) и (iii), должен иметь, по меньшей мере, $2^k + 2^{k-1} - 1$ ребер (связей) между узлами.

*5.4.2. Многофазное слияние

Теперь, после того как мы выяснили, как можно образовать начальные серии, рассмотрим различные методы распределения серий по лентам и их слияния до тех пор, пока не получится единственная серия.

Предположим сначала, что в нашем распоряжении имеются три накопителя на магнитных лентах: T1, T2 и T3; можно воспользоваться методом сбалансированного слияния, описанным в начале раздела 5.4, назначив $P = 2$ и $T = 3$. Алгоритм этого метода принимает следующий вид.

B1. Распределить начальные серии попеременно на ленты T1 и T2.

B2. Выполнить слияние серий с лент T1 и T2 на T3, затем остановиться, если T3 содержит только одну серию.

B3. Скопировать серии с T3 попеременно на T1 и T2, затем вернуться к шагу B2. ▮

Если при начальном распределении получилось S серий, то при первом проходе слияния будет образовано $\lceil S/2 \rceil$ серий на T3, при втором — $\lceil S/4 \rceil$ и т. д. Таким образом, если, скажем, $17 \leq S \leq 32$, то произойдет 1 проход распределения, 5 проходов слияния и 4 прохода копирования. В общем случае, если $S > 1$, число проходов по всем данным будет равно $2\lceil \lg S \rceil$.

Проходы копирования в этой процедуре нежелательны, так как они не уменьшают числа серий. Можно наполовину сократить количество копирований, если использовать *двухфазную* процедуру.

A1. Распределить начальные серии попеременно на ленты T1 и T2.

A2. Слить серии с лент T1 и T2 на T3; остановиться, если на T3 содержится только одна серия.

A3. Скопировать *половину* серий T3 на T1.

A4. Слить серии с лент T1 и T3 на T2; остановиться, если на T2 содержится только одна серия.

A5. Скопировать *половину* серий с T2 на T1. Вернуться к шагу A2. ▮

Число проходов по всем данным сократилось до $\frac{3}{2}\lceil \lg S \rceil + \frac{1}{2}$, так как на шагах A3 и A5 выполняется только "половина прохода", т. е. сэкономлено около 25% времени.

В действительности можно даже *полностью* устранить копирование, если начать с F_n серий на ленте T1 и F_{n-1} серий на T2, где F_n и F_{n-1} — последовательные числа Фибоначчи. Рассмотрим, например, случай, когда $n = 7$, $S = F_n + F_{n-1} = 13 + 8 = 21$.

Фаза	Содержимое T1	Содержимое T2	Содержимое T3	Примечание
1	1,1,1,1,1,1,1,1,1,1,1,1,1	1,1,1,1,1,1,1,1		Начальное распределение
2	1,1,1,1,1	—	2,2,2,2,2,2,2,2	Слияние 8 серий на T3
3	—	3,3,3,3,3	2,2,2	Слияние 5 серий на T2
4	5,5,5	3,3	—	Слияние 3 серий на T1
5	5	—	8,8	Слияние 2 серий на T3
6	—	13	8	Слияние 1 серии на T2
7	21	—	—	Слияние 1 серии на T1

Здесь "2,2,2,2,2,2,2,2", например, обозначает восемь серий относительной длины 2, если считать относительную длину каждой начальной серии равной 1. Всюду в этой таблице присутствуют числа Фибоначчи!

Полный проход по данным осуществляется только на фазах 1 и 7; на фазе 2 обрабатывается лишь 16/21 от общего числа начальных серий, на фазе 3 — лишь 15/21 и т. д. Таким образом, суммарное число "проходов" равно $(21+16+15+15+16+13+21)/21 = 5\frac{4}{7}$, если предположить, что начальные серии имеют примерно равную длину. Для сравнения заметим, что рассмотренная выше двухфазная процедура затратила бы 8 проходов на сортировку этой 21 начальной серии. Мы увидим, что в общем случае данная схема Фибоначчи требует приблизительно $1.04 \lg S + 0.99$ проходов, что делает ее сравнимой с *четырехленточным* сбалансированным слиянием, хотя она использует только три ленты.

Эту идею можно обобщить для T лент при любом $T \geq 3$, используя $(T-1)$-путевое слияние. Мы увидим, например, что для четырех лент требуется только около $.703 \lg S + 0.96$ проходов по данным. Обобщенная схема использует обобщенные числа Фибоначчи. Рассмотрим следующий пример с шестью лентами.

Фаза	T1	T2	T3	T4	T5	T6	Число обрабатываемых начальных серий
1	1^{31}	1^{30}	1^{28}	1^{24}	1^{16}	—	$31 + 30 + 28 + 24 + 16 = 129$
2	1^{15}	1^{14}	1^{12}	1^{8}	—	5^{16}	$16 \times 5 = 80$
3	1^{7}	1^{6}	1^{4}	—	9^{8}	5^{8}	$8 \times 9 = 72$
4	1^{3}	1^{2}	—	17^{4}	9^{4}	5^{4}	$4 \times 17 = 68$
5	1^{1}	—	33^{2}	17^{2}	9^{2}	5^{2}	$2 \times 33 = 66$
6	—	65^{1}	33^{1}	17^{1}	9^{1}	5^{1}	$1 \times 65 = 65$
7	129^{1}	—	—	—	—	—	$1 \times 129 = 129$

Здесь 1^{31} обозначает 31 серию относительной длины 1 и т. д.; везде используется пятипутевое слияние. Эта общая схема была представлена в работе R. L. Gilstad, *Proc. Eastern Joint Computer Conf.* **18** (1960), 143–148, в которой она названа *многофазным слиянием*. Случай для трех лент был открыт Б. К. Бетцем (B. K. Betz) [неопубликованная заметка, Minneapolis-Honeywell Regulator Co. (1956)].

Чтобы заставить механизм многофазного слияния работать, как в предыдущем примере, необходимо после каждой фазы иметь "точное фибоначчиево распреде-

ление" серий по лентам. Читая приведенную выше таблицу снизу вверх, можно заметить, что первые семь точных фибоначчиевых распределений при $T = 6$ суть $\{1,0,0,0,0\}$, $\{1,1,1,1,1\}$, $\{2,2,2,2,1\}$, $\{4,4,4,3,2\}$, $\{8,8,7,6,4\}$, $\{16,15,14,12,8\}$ и $\{31,30,28,24,16\}$. Теперь перед нами стоят следующие важные вопросы.

1. Какое правило скрыто за этими точными фибоначчиевыми распределениями?
2. Что делать, если S не соответствует фибоначчиевому распределению?
3. Как построить начальный проход распределения, чтобы на нем порождалось нужное расположение серий на лентах?
4. Сколько "проходов" по данным потребует T-ленточное многофазное слияние (как функция от S — числа начальных серий)?

Мы обсудим эти четыре вопроса по очереди; сначала дадим "простые ответы", а затем займемся более глубоким анализом.

Точные фибоначчиевы распределения можно получить, "прокручивая" рассмотренную схему в обратном направлении и циклически переставляя содержимое лент. Например, при $T = 6$ имеем следующее распределение серий.

Уровень	T1	T2	T3	T4	T5	Сумма	Окончательный результат будет на
0	1	0	0	0	0	1	T1
1	1	1	1	1	1	5	T6
2	2	2	2	2	1	9	T5
3	4	4	4	3	2	17	T4
4	8	8	7	6	4	33	T3
5	16	15	14	12	8	65	T2
6	31	30	28	24	16	129	T1
7	61	59	55	47	31	253	T6
8	120	116	108	92	61	497	T5
. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .
n	a_n	b_n	c_n	d_n	e_n	t_n	T(k)
$n+1$	$a_n + b_n$	$a_n + c_n$	$a_n + d_n$	$a_n + e_n$	a_n	$t_n + 4a_n$	T($k-1$) (1)
. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .

(После начального распределения лента T6 всегда будет пустой.)

Из правила перехода от уровня n к уровню $n+1$ ясно, что условия

$$a_n \geq b_n \geq c_n \geq d_n \geq e_n \tag{2}$$

выполняются на любом уровне. В самом деле, легко видеть из (1), что

$$\begin{aligned}
e_n &= a_{n-1}, \\
d_n &= a_{n-1} + e_{n-1} = a_{n-1} + a_{n-2}, \\
c_n &= a_{n-1} + d_{n-1} = a_{n-1} + a_{n-2} + a_{n-3}, \\
b_n &= a_{n-1} + c_{n-1} = a_{n-1} + a_{n-2} + a_{n-3} + a_{n-4}, \\
a_n &= a_{n-1} + b_{n-1} = a_{n-1} + a_{n-2} + a_{n-3} + a_{n-4} + a_{n-5},
\end{aligned} \tag{3}$$

где $a_0 = 1$ и где мы полагаем, что $a_n = 0$ при $n = -1, -2, -3, -4$.

Числа Фибоначчи p-го порядка $F_n^{(p)}$ определяются правилами

$$\begin{aligned} F_n^{(p)} &= F_{n-1}^{(p)} + F_{n-2}^{(p)} + \cdots + F_{n-p}^{(p)} \quad \text{при } n \geq p; \\ F_n^{(p)} &= 0 \quad \text{при } 0 \leq n \leq p-2; \qquad F_{p-1}^{(p)} = 1. \end{aligned} \tag{4}$$

Другими словами, мы начинаем с $p-1$ нулей, затем пишем 1 и каждое следующее число является суммой p предыдущих чисел. При $p = 2$ это обычная последовательность Фибоначчи; для больших значений p ее впервые изучил, по-видимому, В. Шлегель (V. Schlegel) в *El Progreso Matemático* **4** (1894), 173–174. Шлегель вывел производящую функцию

$$\sum_{n\geq 0} F_n^{(p)} z^n = \frac{z^{p-1}}{1 - z - z^2 - \cdots - z^p} = \frac{z^{p-1} - z^p}{1 - 2z + z^{p+1}}. \tag{5}$$

Формула (3) показывает, что число серий на T1 в процессе шестиленточного многофазного слияния является числом Фибоначчи пятого порядка: $a_n = F_{n+4}^{(5)}$.

В общем случае, если положить $P = T - 1$, распределения в многофазном слиянии для T лент будут аналогичным образом соответствовать числам Фибоначчи P-го порядка. В точном распределении n-го уровня на k-й ленте будет

$$F_{n+P-2}^{(P)} + F_{n+P-3}^{(P)} + \cdots + F_{n+k-2}^{(P)}$$

начальных серий для $1 \leq k \leq P$, а общее количество начальных серий на всех лентах будет, следовательно, равно

$$t_n = PF_{n+P-2}^{(P)} + (P-1)F_{n+P-3}^{(P)} + \cdots + F_{n-1}^{(P)}. \tag{6}$$

Это решает вопрос о "точном фибоначчиевом распределении". Но что мы должны делать, если S не равно в точности t_n ни при каком n? Как первоначально поместить серии на ленты?

Если S не является точным числом Фибоначчи (а чисел Фибоначчи не так уж много), то можно действовать, как в сбалансированном P-путевом слиянии, добавляя "фиктивные серии"; поэтому можно считать, что S, в конце концов, будет точным. Существует несколько способов добавления фиктивных серий, но мы еще не знаем, как это сделать наилучшим образом. В первую очередь, рассмотрим метод распределения серий и приписывания фиктивных серий, который хотя и не самый оптимальный, но зато достаточно простой и, по-видимому, лучше всех остальных методов такой же степени сложности.

Алгоритм D (*Сортировка методом многофазного слияния с использованием "горизонтального" распределения*). Этот алгоритм (рис. 68) берет начальные серии и распределяет их одну за другой по лентам, пока запас начальных серий не исчерпается. Затем он определяет, как необходимо сливать ленты, используя P-путевое слияние, в предположении, что имеются $T = P + 1 \geq 3$ накопителей на магнитной ленте. Ленту T можно применять для хранения исходных данных, так как на нее не записывается ни одна начальная серия. В памяти хранятся следующие таблицы.

`A[j]`, $1 \leq j \leq T$: Точное фибоначчиево распределение, к которому мы стремимся

`D[j]`, $1 \leq j \leq T$: Число фиктивных серий, которые считаются присутствующими в начале ленты на логическом устройстве с номером j

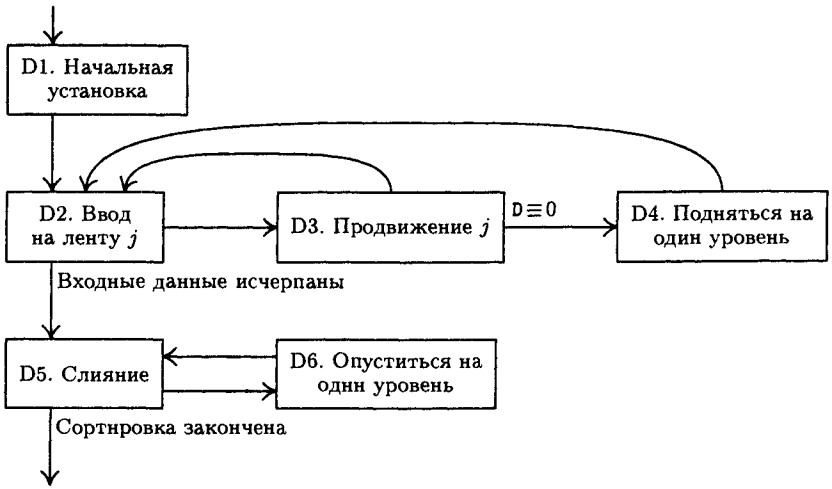


Рис. 68. Сортировка методом многофазного слияния.

TAPE[j], $1 \le j \le T$: Номер физического накопителя на магнитной ленте, соответствующего логическому устройству с номером j

(Удобно работать с номерами "логических" накопителей на магнитной ленте, соответствие которых физическим накопителям меняется в процессе выполнения алгоритма.)

D1. [Начальная установка.] Установить A[j] $\leftarrow$ D[j] $\leftarrow 1$ и TAPE[j] $\leftarrow j$ при $1 \le j < T$. Установить A[T] $\leftarrow$ D[T] $\leftarrow 0$ и TAPE[T] $\leftarrow T$. Затем установить $l \leftarrow 1$, $j \leftarrow 1$.

D2. [Ввод на ленту j.] Записать одну серию на ленту j и уменьшить D[j] на 1. Затем, если ввод исчерпан, перемотать все ленты и перейти к шагу D5.

D3. [Продвижение j.] Если D[j] $<$ D[$j+1$], то увеличить j на 1 и вернуться к шагу D2. В противном случае, если D[j] $= 0$, перейти к шагу D4, иначе — установить $j \leftarrow 1$ и вернуться к шагу D2.

D4. [Подняться на один уровень.] Установить $l \leftarrow l+1$, $a \leftarrow$ A[1], затем для $j = 1, 2, \ldots, P$ (именно в таком порядке) установить D[j] $\leftarrow a +$ A[$j+1$] $-$ A[j] и A[j] $\leftarrow a +$ A[$j+1$]. (См. (1). Отметим, что A[$P+1$] всегда 0. В этом месте будем иметь D[1] $\ge$ D[2] $\ge \cdots \ge$ D[T].) Теперь установить $j \leftarrow 1$ и вернуться к шагу D2.

D5. [Слияние.] Если $l = 0$, то сортировка завершена и результат находится на TAPE[1]. В противном случае выполнять слияние серий с лент TAPE[1], ..., TAPE[P] на TAPE[T] до тех пор, пока TAPE[P] не станет пустой и D[P] $= 0$. Процесс слияния для каждой сливаемой серии должен протекать следующим образом. Если D[j] > 0 для всех j, $1 \le j \le P$, то увеличить D[T] на 1 и уменьшить каждое D[j] на 1 для $1 \le j \le P$; в противном случае выполнять слияние по одной серии с каждой ленты TAPE[j], такой, что D[j] $= 0$, и уменьшить D[j] на 1 для остальных j. (Таким образом, считается, что фиктивные серии находятся в *начале* ленты, а не в конце.)

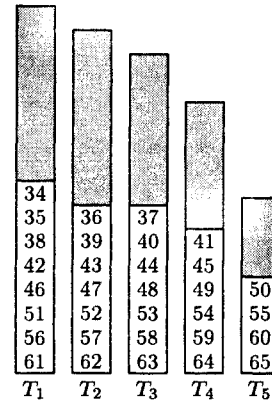


Рис. 69. Порядок, в котором серии 34–65 распределяются на ленты при переходе с уровня 4 на уровень 5. (См. таблицу точных распределений (1).) Заштрихованные области соответствуют первым 33 сериям, которые были распределены к моменту достижения уровня 4. Последняя строка в каждой колонке соответствует началу ленты.

D6. [Опуститься на один уровень.] Установить $l \leftarrow l-1$. Перемотать ленты TAPE[P] и TAPE[T]. (В действительности перемотка TAPE[P] могла быть начата на шаге D5, после ввода с нее последнего блока.) Затем установить (TAPE[1], TAPE[2], ..., TAPE[T]) $\leftarrow$ (TAPE[T], TAPE[1], ..., TAPE[$T-1$]), (D[1], D[2], ..., D[T]) $\leftarrow$ (D[T], D[1], ..., D[$T-1$]) и вернуться к шагу D5. ▌

Правило распределения, которое так лаконично сформулировано на шаге D3 этого алгоритма, стремится по возможности уравнять числа фиктивных серий на каждой ленте. На рис. 69 иллюстрируется распределение серий, когда мы переходим от уровня 4 (33 серии) к уровню 5 (65 серий) в сортировке с шестью лентами; если было бы, скажем, только 53 начальные серии, то все серии с номерами 54 и выше рассматривались бы как фиктивные. (На самом деле серии записываются в конец ленты, но удобнее считать, что они записываются в начало, так как предполагается, что фиктивные серии находятся в начале ленты.)

Мы уже рассмотрели первые три из поставленных выше вопросов; осталось выяснить число "проходов" по данным. Сравнивая наш "шестиленточный пример" с таблицей (1), мы видим, что суммарное количество обработанных начальных серий при $S = t_6$ составляет $a_5t_1+a_4t_2+a_3t_3+a_2t_4+a_1t_5+a_0t_6$, если исключить начальный проход распределения. В упр. 4 выводятся производящие функции

$$\begin{aligned} a(z) &= \sum_{n\ge 0} a_n z^n = \frac{1}{1-z-z^2-z^3-z^4-z^5}, \\ t(z) &= \sum_{n\ge 1} t_n z^n = \frac{5z+4z^2+3z^3+2z^4+z^5}{1-z-z^2-z^3-z^4-z^5}. \end{aligned} \tag{7}$$

Отсюда следует, что в общем случае число обрабатываемых начальных серий при $S = t_n$ равно коэффициенту при z^n в произведении $a(z)t(z)$ плюс t_n (это дает начальный проход распределения). Теперь вычисляем асимптотическое поведение многофазного слияния, как показано в упр. 5–7, и получаем результаты, приведенные в табл. 1.

В табл. 1 "Относительный рост" есть предел $\lim_{n\to\infty} t_{n+1}/t_n$, показывающий, во сколько приблизительно раз возрастает число серий на каждом проходе. "Проходы" — это среднее количество обработок каждой записи, а именно — $1/S$, умноженное на общее число начальных серий, обрабатываемых в течение фаз распреде-

Таблица 1

АППРОКСИМАЦИЯ ПОВЕДЕНИЯ СОРТИРОВКИ МЕТОДОМ МНОГОФАЗНОГО СЛИЯНИЯ

Ленты	Фазы	Проходы	Проходы/ фазы, %	Относительный рост
3	$2.078 \ln S + 0.672$	$1.504 \ln S + 0.992$	72	1.6180340
4	$1.641 \ln S + 0.364$	$1.015 \ln S + 0.965$	62	1.8392868
5	$1.524 \ln S + 0.078$	$0.863 \ln S + 0.921$	57	1.9275620
6	$1.479 \ln S - 0.185$	$0.795 \ln S + 0.864$	54	1.9659482
7	$1.460 \ln S - 0.424$	$0.762 \ln S + 0.797$	52	1.9835828
8	$1.451 \ln S - 0.642$	$0.744 \ln S + 0.723$	51	1.9919642
9	$1.447 \ln S - 0.838$	$0.734 \ln S + 0.646$	51	1.9960312
10	$1.445 \ln S - 1.017$	$0.728 \ln S + 0.568$	50	1.9980295
20	$1.443 \ln S - 2.170$	$0.721 \ln S - 0.030$	50	1.9999981

ления и слияния. Указанные числа проходов и фаз справедливы в каждом случае с точностью до $O(S^{-\epsilon})$ при некотором $\epsilon > 0$ для точного распределения при $S \to \infty$.

На рис. 70 изображены в виде функций от S средние количества слияний каждой записи с помощью алгоритма D для неточных чисел. Заметим, что при использовании трех лент сразу после точных распределений появляются "пики" относительной неэффективности, но это явление в значительной степени исчезает при наличии четырех или более лент. Использование восьми или более лент дает сравнительно малое улучшение по сравнению с шестью или семью лентами.

Более детальный анализ. В сбалансированном слиянии, требующем k проходов, каждая запись обрабатывается в ходе сортировки ровно k раз. Но многофазная процедура не является такой беспристрастной: одни записи могут обрабатываться намного больше раз, чем другие, и можно увеличить скорость, если помещать фиктивные серии в часто обрабатываемые позиции.

По этой причине проанализируем более подробно многофазное распределение. Вместо того чтобы интересоваться только числом серий на каждой ленте, как в (1), присвоим каждой серии *число слияний* , т. е. определим, сколько раз она будет обрабатываться в течение всего процесса сортировки. Вместо (1) получим следующую таблицу.

Уровень	T1	T2	T3	T4	T5
0	0	—	—	—	—
1	1	1	1	1	1
2	21	21	21	21	2
3	3221	3221	3221	322	32
4	43323221	43323221	4332322	433232	4332
5	5443433243323221	544343324332322	54434332433232	544343324332	54434332
. . .	. . .	. . .	. . .	. . .	. . .
n	A_n	B_n	C_n	D_n	E_n
$n+1$	$(A_n+1)B_n$	$(A_n+1)C_n$	$(A_n+1)D_n$	$(A_n+1)E_n$	A_n+1
. . .	. . .	. . .	. . .	. . .	. . .

(8)

Здесь A_n — цепочка из a_n значений, представляющих числа слияний каждой серии на T1, если начать с распределения n-го уровня; B_n — соответствующая цепочка для T2 и т. д. Обозначение "$(A_n + 1)B_n$" читается так: "A_n, все значения которой увеличены на 1, а за ней — B_n".

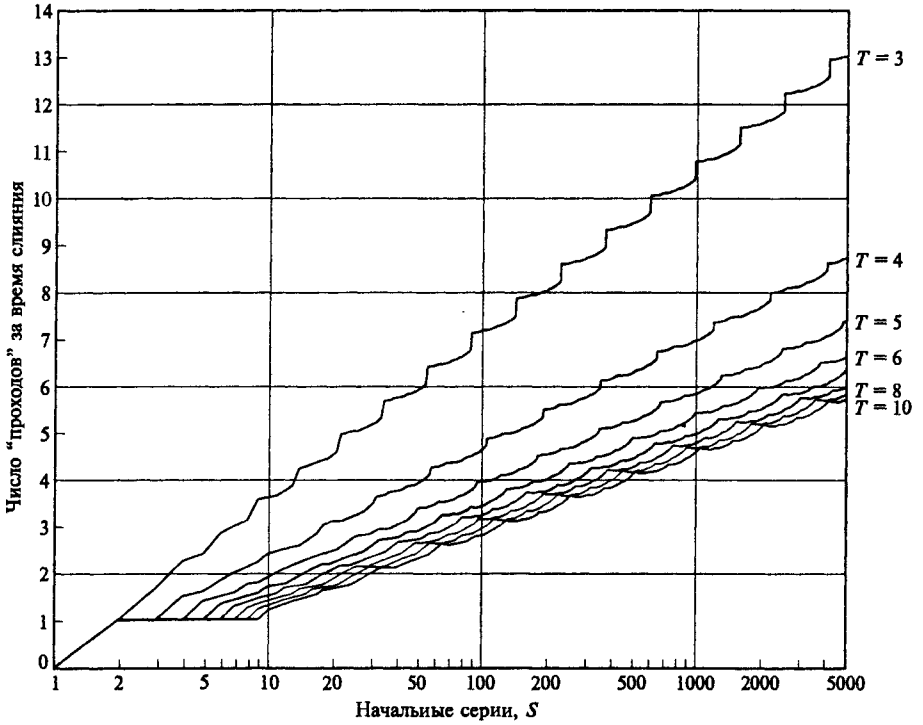


Рис. 70. Эффективность многофазного слияния, использующего алгоритм D.

На рис. 71, (a) изображаются снизу вверх A_5, B_5, C_5, D_5, E_5 и демонстрируется, каким образом числа слияний для каждой серии появляются на ленте. Заметим, что серия в начале любой ленты будет обрабатываться 5 раз, в то время как серия в конце T1 будет обрабатываться лишь однажды. Эта "дискриминация" при многофазном слиянии приводит к тому, что фиктивные серии лучше помещать в начало ленты, а не в конец. На рис. 71, (b) представлен оптимальный порядок распределения серий для пятиуровневого многофазного слияния; каждая новая серия помещается в позицию с наименьшим из оставшихся числом слияний. Заметим, что алгоритм D (см. рис. 69) несколько хуже, так как он заполняет некоторые позиции "4" до того, как будут заполнены все позиции "3".

Рекуррентные соотношения (8) показывают, что все B_n, C_n, D_n и E_n являются начальными подцепочками A_n. В действительности, используя (8), можно вывести формулы

$$\begin{aligned}
E_n &= (A_{n-1}) + 1,\\
D_n &= (A_{n-1}A_{n-2}) + 1,\\
C_n &= (A_{n-1}A_{n-2}A_{n-3}) + 1,\\
B_n &= (A_{n-1}A_{n-2}A_{n-3}A_{n-4}) + 1,\\
A_n &= (A_{n-1}A_{n-2}A_{n-3}A_{n-4}A_{n-5}) + 1,
\end{aligned} \tag{9}$$

1	2	2	2	2		1	3	6	10	15	
2	2	3	3	3		2	5	18	22	31	
2	3	2	3	3		4	17	9	26	36	
3	2	3	3	4		16	8	21	45	50	
2	3	3	4	3		7	20	25	14	41	
3	3	4	4	4		19	24	44	30	55	
3	4	2	2	4		23	43	13	35	60	
4	2	3	3	3		42	12	29	49	65	
2	3	3	4	4		11	28	34	40		
3	3	4	3	3		27	33	48	54		
3	4	3	4	4		32	47	39	59		
4	3	4	4	4		46	38	53	64		
4	4	4	4	4		37	52	58			
5	5	5	5	5	Начало ленты	51	57	63			
						56	62				
						61					
(a)						(b)					

Рис. 71. Анализ многофазного распределения пятого уровня на шести лентах: (a) — числа слияний; (b) — оптимальный порядок распределения.

обобщающие соотношения (3), которые имеют дело только с длинами этих цепочек. Кроме того, из правил, определяющих цепочки A, следует, что структура в начале каждого уровня, в сущности, одна и та же; имеем

$$A_n = n - Q_n, \tag{10}$$

где Q_n — цепочка из a_n значений, определяемая законом

$$\begin{aligned} &Q_n = Q_{n-1}(Q_{n-2}+1)(Q_{n-3}+2)(Q_{n-4}+3)(Q_{n-5}+4) \quad \text{при } n \geq 1; \\ &Q_0 = 0; \qquad\qquad Q_n = \epsilon \quad \text{(пустая цепочка)} \quad \text{при } n < 0. \end{aligned} \tag{11}$$

Так как Q_n начинается с Q_{n-1}, можно рассмотреть *бесконечную* цепочку Q_∞, первые a_n элементов которой совпадают с Q_n. Эта цепочка Q_∞, по существу, описывает все числа слияний в многофазном распределении. В случае шести лент имеем

$$Q_\infty = 011212231223233412232334233434412232334233434452334344534454512232\cdots. \tag{12}$$

В упр. 11 содержится интересная интерпретация этой цепочки.

При условии, что A_n есть цепочки $m_1m_2\ldots m_{a_n}$, обозначим через $A_n(x) = x^{m_1} + x^{m_2} + \cdots + x^{m_{a_n}}$ соответствующую производящую функцию, описывающую, сколько раз появляется каждое число слияний; аналогично введем $B_n(x)$, $C_n(x)$, $D_n(x)$, $E_n(x)$. Например, $A_4(x) = x^4+x^3+x^3+x^2+x^3+x^2+x^2+x = x^4+3x^3+3x^2+x$. В силу соотношений (9) при $n \geq 1$ имеем

$$\begin{aligned} E_n(x) &= x\big(A_{n-1}(x)\big), \\ D_n(x) &= x\big(A_{n-1}(x) + A_{n-2}(x)\big), \\ C_n(x) &= x\big(A_{n-1}(x) + A_{n-2}(x) + A_{n-3}(x)\big), \\ B_n(x) &= x\big(A_{n-1}(x) + A_{n-2}(x) + A_{n-3}(x) + A_{n-4}(x)\big), \\ A_n(x) &= x\big(A_{n-1}(x) + A_{n-2}(x) + A_{n-3}(x) + A_{n-4}(x) + A_{n-5}(x)\big), \end{aligned} \tag{13}$$

где $A_0(x) = 1$ и $A_n(x) = 0$ при $n = -1, -2, -3, -4$. Следовательно,

$$\sum_{n\geq 0} A_n(x)z^n = \frac{1}{1-x(z+z^2+z^3+z^4+z^5)} = \sum_{k\geq 0} x^k(z+z^2+z^3+z^4+z^5)^k. \qquad (14)$$

Рассматривая серии на всех лентах, положим

$$T_n(x) = A_n(x) + B_n(x) + C_n(x) + D_n(x) + E_n(x), \qquad n \geq 1; \qquad (15)$$

из (13) немедленно получаем

$$T_n(x) = 5A_{n-1}(x) + 4A_{n-2}(x) + 3A_{n-3}(x) + 2A_{n-4}(x) + A_{n-5}(x),$$

а значит, и

$$\sum_{n\geq 1} T_n(x)z^n = \frac{x(5z+4z^2+3z^3+2z^4+z^5)}{1-x(z+z^2+z^3+z^4+z^5)}. \qquad (16)$$

Соотношение (16) показывает, что можно легко вычислить коэффициенты $T_n(x)$.

	z	z^2	z^3	z^4	z^5	z^6	z^7	z^8	z^9	z^{10}	z^{11}	z^{12}	z^{13}	z^{14}
x	5	4	3	2	1	0	0	0	0	0	0	0	0	0
x^2	0	5	9	12	14	15	10	6	3	1	0	0	0	0
x^3	0	0	5	14	26	40	55	60	57	48	35	20	10	4
x^4	0	0	0	5	19	45	85	140	195	238	260	255	220	170
x^5	0	0	0	0	5	24	69	154	294	484	703	918	1088	1168

(17)

Столбцы этой таблицы дают $T_n(x)$; например, $T_4(x) = 2x + 12x^2 + 14x^3 + 5x^4$. Каждый элемент данной таблицы (кроме элементов первой строки) является суммой пяти элементов, расположенных в предыдущей строке непосредственно левее него.

Число серий в "точном" распределении n-го уровня равно $T_n(1)$, а общее количество обрабатываемых серий в процессе их слияния равно производной $T_n'(1)$. Далее,

$$\sum_{n\geq 1} T_n'(x)z^n = \frac{5z+4z^2+3z^3+2z^4+z^5}{\left(1-x(z+z^2+z^3+z^4+z^5)\right)^2}; \qquad (18)$$

полагая $x = 1$ в (16) и (18), получаем, что число слияний для точного распределения n-го уровня есть коэффициент при z^n в $a(z)t(z)$; см. (7). Это согласуется с нашими предыдущими рассуждениями.

Функции $T_n(x)$ можно использовать для определения объема работы, когда фиктивные серии добавляются оптимальным образом. Пусть $\Sigma_n(m)$ есть сумма наименьших m чисел слияний в распределении n-го уровня. Посмотрев на столбцы (17), мы без труда вычислим эти суммы $\Sigma_n(m)$.

$m =$	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
$n=1$	1	2	3	4	5	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
$n=2$	1	2	3	4	6	8	10	12	14	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
$n=3$	1	2	3	5	7	9	11	13	15	17	19	21	24	27	30	33	36	∞	∞	∞	∞
$n=4$	1	2	4	6	8	10	12	14	16	18	20	22	24	26	29	32	35	38	41	44	47
$n=5$	1	3	5	7	9	11	13	15	17	19	21	23	25	27	29	32	35	38	41	44	47
$n=6$	2	4	6	8	10	12	14	16	18	20	22	24	26	28	30	33	36	39	42	45	48
$n=7$	2	4	6	8	10	12	14	16	18	20	23	26	29	32	35	38	41	44	47	50	53

(19)

Например, для сортировки 17 серий с помощью распределения 3-го уровня общее количество операций составит $\Sigma_3(17) = 36$. Но, если использовать распределение

Таблица 2

ЧИСЛО СЕРИЙ, ПРИ КОТОРОМ ДАННЫЙ УРОВЕНЬ ОПТИМАЛЕН

Уровень	$T=3$	$T=4$	$T=5$	$T=6$	$T=7$	$T=8$	$T=9$	$T=10$	
1	2	2	2	2	2	2	2	2	M_1
2	3	4	5	6	7	8	9	10	M_2
3	4	6	8	10	12	14	16	18	M_3
4	6	10	14	14	17	20	23	26	M_4
5	9	18	23	29	20	24	28	32	M_5
6	14	32	35	43	53	27	32	37	M_6
7	22	55	76	61	73	88	35	41	M_7
8	35	96	109	154	98	115	136	44	M_8
9	56	173	244	216	283	148	171	199	M_9
10	90	280	359	269	386	168	213	243	M_{10}
11	145	535	456	779	481	640	240	295	M_{11}
12	234	820	1197	1034	555	792	1002	330	M_{12}
13	378	1635	1563	1249	1996	922	1228	1499	M_{13}
14	611	2401	4034	3910	2486	1017	1432	1818	M_{14}
15	988	4959	5379	4970	2901	4397	1598	2116	M_{15}
16	1598	7029	6456	5841	10578	5251	1713	2374	M_{16}
17	2574	14953	18561	19409	13097	5979	8683	2576	M_{17}
18	3955	20583	22876	23918	15336	6499	10069	2709	M_{18}
19	6528	44899	64189	27557	17029	30164	11259	15787	M_{19}

4- или 5-го уровня, общее количество операций в процессе слияния будет только $\Sigma_4(17) = \Sigma_5(17) = 35$. Выгоднее применять 4-й уровень, хотя число 17 соответствует "точному" распределению 3-го уровня! В самом деле, по мере возрастании S оптимальный номер уровня оказывается значительно больше номера, используемого в алгоритме D.

В упр. 14 показано, что существует неубывающая последовательность M_n, такая, что уровень n оптимален для $M_n \le S < M_{n+1}$, но не для $S \ge M_{n+1}$. В случае для шести лент только что вычисленная таблица $\Sigma_n(m)$ дает

$$M_0 = 0, \qquad M_1 = 2, \qquad M_2 = 6, \qquad M_3 = 10, \qquad M_4 = 14.$$

Выше рассматривался только случай для шести лент, однако ясно, что те же идеи применимы к многофазной сортировке с T лентами для любого $T \ge 3$; просто в соответствующих местах необходимо заменить 5 на $P = T - 1$. В табл. 2 представлены последовательности M_n, полученные для различных значений T. Табл. 3 и рис. 72 дают представление об общем количестве обрабатываемых начальных серий после оптимального распределения фиктивных серий. (Формулы внизу табл. 3 следует принимать с осторожностью, так как это приближение по методу наименьших квадратов на области $1 \le S \le 5000$ ($1 \le S \le 10000$ для $T = 3$), что приводит к отклонению (данная область значений S не является одинаково подходящей для всех T). По мере того, как $S \to \infty$, число обрабатываемых начальных серий после оптимального многофазного распределения асимптотически приближается к $S \log_P S$, но это приближение происходит крайне медленно.)

При помощи табл. 4 можно сравнить метод распределения алгоритма D с результатами оптимального распределения, приведенными в табл. 3. Ясно, что алгоритм D

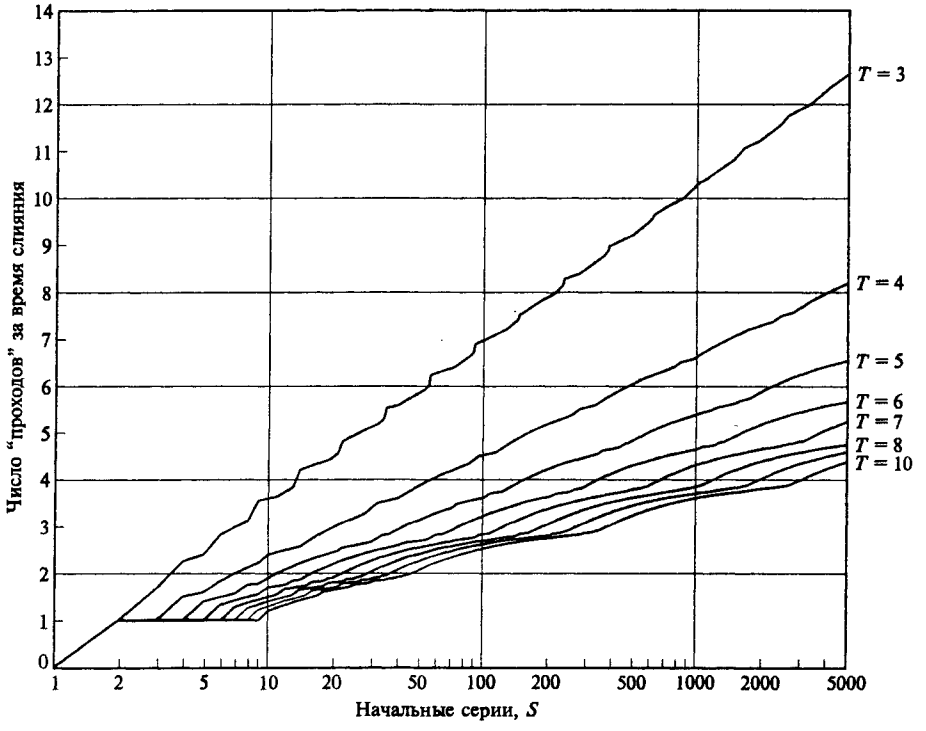


Рис. 72. Эффективность многофазного слияния с оптимальным начальным распределением при тех же предположениях, что и на рис. 70.

Таблица 3

ЧИСЛО НАЧАЛЬНЫХ СЕРИЙ, ОБРАБАТЫВАЕМЫХ ПРИ ОПТИМАЛЬНОМ МНОГОФАЗНОМ СЛИЯНИИ

S	$T=3$	$T=4$	$T=5$	$T=6$	$T=7$	$T=8$	$T=9$	$T=10$
10	36	24	19	17	15	14	13	12
20	90	60	49	44	38	36	34	33
50	294	194	158	135	128	121	113	104
100	702	454	362	325	285	271	263	254
500	4641	3041	2430	2163	1904	1816	1734	1632
1000	10371	6680	5430	4672	4347	3872	3739	3632
5000	63578	41286	32905	28620	26426	23880	23114	22073
S	$\begin{cases}(1.51 \\ (-.11\end{cases}$	0.951 +.14	0.761 +.16	0.656 +.19	0.589 +.21	0.548 +.20	0.539 +.02	0.488) $\times S \ln S +$ +.18) $\times S$

не очень близок к оптимальному при больших S и T; однако непонятно, можно ли поступить в этих случаях существенно лучше алгоритма D, не прибегая к значительным усложнениям, особенно если S заранее неизвестно. К счастью, заботиться о больших S приходится довольно редко (см. раздел 5.4.6), так что алгоритм D на практике не так уж плох (а на самом деле — даже весьма неплох!).

Таблица 4

ЧИСЛО НАЧАЛЬНЫХ СЕРИЙ, ОБРАБАТЫВАЕМЫХ ПРИ СТАНДАРТНОМ МНОГОФАЗНОМ СЛИЯНИИ

S	$T=3$	$T=4$	$T=5$	$T=6$	$T=7$	$T=8$	$T=9$	$T=10$
10	36	24	19	17	15	14	13	12
20	90	62	49	44	41	37	34	33
50	294	194	167	143	134	131	120	114
100	714	459	393	339	319	312	292	277
500	4708	3114	2599	2416	2191	2100	2047	2025
1000	10730	6920	5774	5370	4913	4716	4597	4552
5000	64740	43210	36497	32781	31442	29533	28817	28080

Математически многофазная сортировка впервые проанализирована У. К. Картером (W. C. Carter) [*Proc. IFIP Congress* (1962), 62–66]. Многие из приведенных им результатов относительно оптимального размещения принадлежат Б. Сэкману (B. Sackman) и Т. Зингеру (T. Singer) ["A vector model for merge sort analysis", неопубликованный доклад, представленный на симпозиуме ACM Sort Symposium (November, 1962), 21 pages]. Позднее Сэкман предложил метод распределения по горизонтали, используемый в алгоритме D. Дональд Шелл (Donald Shell) [*CACM* **14** (1971), 713–719; **15** (1972), 28], независимо развив эту теорию, указал на соотношение (10) и подробно изучил несколько различных алгоритмов распределения. Дальнейшие полезные усовершенствования и упрощения были получены Дереком Э. Зэйвом (Derek A. Zave) [*SICOMP* **6** (1977), 1–39]; некоторые из результатов Зэйва рассматриваются в упр. 15–17. Производящая функция (16) впервые была исследована У. Буржем (W. Burge) [*Proc. IFIP Congress* (1971), **1**, 454–459].

А как обстоит дело с временем перемотки? До сих пор мы использовали число начальных обрабатываемых серий как единственную меру эффективности для сравнения стратегий слияния на лентах. Но после каждой из фаз 2–6 в примерах в начале этого раздела компьютер должен ожидать перемотки двух лент; обе выводные ленты — как предыдущая, так и новая текущая — должны быть перемотаны в начало, прежде чем сможет выполняться следующая фаза. Это может вызвать существенную задержку, так как в общем случае на предыдущей выводной ленте содержится значительный процент сортируемых записей (см. столбец "Проходы/фазы" в табл. 1). Досадно, когда компьютер простаивает при перемотке, тогда как в это время можно было бы, используя иную схему слияния, выполнять полезную работу с остальными лентами.

Данную задачу можно решить с помощью простой модификации многофазной процедуры, хотя она требует не менее пяти лент [см. диссертацию И. Сезари (Y. Césari) (U. of Paris (1968), 25–27), в которой эта идея приписывается Ж. Карону (J. Caron)]. Каждая фаза схемы Карона сливает серии с $T-3$ лент на любую другую ленту, в то время как оставшиеся две ленты перематываются.

Рассмотрим, например, случай для шести лент и 49 начальных серий. В следующей таблице буквой R обозначены ленты, перематывающиеся во время данной фазы; предполагается, что на ленте T5 содержатся первоначальные серии.

Фаза	T1	T2	T3	T4	T5	T6	Время записи	Время перемотки
1	1^{11}	1^{17}	1^{13}	1^{8}	—	(R)	49	17
2	(R)	1^{9}	1^{5}	—	R	3^{8}	$8 \times 3 = 24$	$49 - 17 = 32$
3	1^{6}	1^{4}	—	R	3^{5}	R	$5 \times 3 = 15$	$\max(8, 24)$
4	1^{2}	—	R	5^{4}	R	3^{4}	$4 \times 5 = 20$	$\max(13, 15)$
5	—	R	7^{2}	R	3^{3}	3^{2}	$2 \times 7 = 14$	$\max(17, 20)$
6	R	11^{2}	R	5^{2}	3^{1}	—	$2 \times 11 = 22$	$\max(11, 14)$
7	15^{1}	R	7^{1}	5^{1}	—	R	$1 \times 15 = 15$	$\max(22, 24)$
8	R	11^{1}	7^{0}	—	R	23^{1}	$1 \times 23 = 23$	$\max(15, 15)$
9	15^{1}	11^{1}	—	R	33^{0}	R	$0 \times 33 = 0$	$\max(20, 23)$
10	(15^{0})	—	R	49^{1}	(R)	(23^{0})	$1 \times 49 = 49$	14

Здесь все перемотки, по существу, совмещены, за исключением фазы 9 ("фиктивная фаза", которая подготавливает окончательное слияние) и перемотки после начальной фазы распределения (когда перематываются все ленты). Если t есть время, необходимое для слияния стольких записей, сколько содержится в одной начальной серии, а r — время перемотки на одну начальную серию, то этому процессу необходимо около $182t + 40r$ плюс время начального распределения и завершающей перемотки. Соответствующие выражения для стандартного многофазного метода, использующего алгоритм D, есть $140t + 104r$, что несколько хуже, если $r = \frac{3}{4}t$, и несколько лучше, если $r = \frac{1}{2}t$.

Все сказанное о стандартном многофазном методе приложимо к многофазному методу Карона; например, последовательность a_n теперь удовлетворяет рекуррентному соотношению

$$a_n = a_{n-2} + a_{n-3} + a_{n-4} \tag{20}$$

вместо (3). Читателю будет полезно проанализировать этот метод таким же образом, как мы анализировали стандартный многофазный, поскольку это улучшит понимание обоих методов (см., например, упр. 19 и 20).

В табл. 5 сведены данные о многофазном методе Карона, аналогичные данным об обычном многофазном методе, приведенным в табл. 1. Заметим, что на самом деле при восьми и более лентах метод Карона становится *лучше* многофазного как по числу обрабатываемых серий, так и по времени перемотки, несмотря на то что он выполняет $(T-3)$-путевое слияние вместо $(T-1)$-путевого!

Таблица 5

ПРИБЛИЗИТЕЛЬНЫЕ ДАННЫЕ О ХОДЕ СОРТИРОВКИ МЕТОДОМ МНОГОФАЗНОГО СЛИЯНИЯ КАРОНА

Ленты	Фазы	Проходы	Проходы/ фазы, %	Отношение роста
5	$3.556 \ln S + 0.158$	$1.463 \ln S + 1.016$	41	1.3247180
6	$2.616 \ln S - 0.166$	$0.951 \ln S + 1.014$	36	1.4655712
7	$2.337 \ln S - 0.472$	$0.781 \ln S + 1.001$	33	1.5341577
8	$2.216 \ln S - 0.762$	$0.699 \ln S + 0.980$	32	1.5701473
9	$2.156 \ln S - 1.034$	$0.654 \ln S + 0.954$	30	1.5900054
10	$2.124 \ln S - 1.290$	$0.626 \ln S + 0.922$	29	1.6013473
20	$2.078 \ln S - 3.093$	$0.575 \ln S + 0.524$	28	1.6179086

Эти данные будут казаться парадоксальными, пока мы не поймем, что *высокий порядок слияния не обязательно означает эффективную сортировку*. В качестве крайнего рассмотрим случай, когда 1 серия помещается на ленту T1 и n серий — на T2, T3, T4, T5; если поочередно выполнять слияния на T6 и T1, пока T2, T3, T4, T5 не станут пустыми, то время обработки будет соответствовать $(2n^2 + 3n)$ длинам начальных серий, т. е., по существу, будет пропорционально S^2, а не $S \log S$, хотя все время производится пятипутевое слияние.

Расщепление лент. Эффективное совмещение времени перемотки является проблемой, возникающей во многих приложениях, а не только при сортировке. Существует общий подход, который можно использовать в большинстве случаев. Рассмотрим итеративный процесс, в котором ленты используются следующим образом.

	T1	T2
Фаза 1	Вывод 1	—
	Перемотка	—
Фаза 2	Ввод 1	Вывод 2
	Перемотка	Перемотка
Фаза 3	Вывод 3	Ввод 2
	Перемотка	Перемотка
Фаза 4	Ввод 3	Вывод 4
	Перемотка	Перемотка

и т. д. Здесь "Вывод k" означает запись в k-й выводной файл, а "Ввод k" — его чтение. Можно устранить время перемотки, если использовать три ленты, как было предложено в работе L. Weisner, *CACM* **5** (1962), 102:

	T1	T2	T3
Фаза 1	Вывод 1.1	—	—
	Вывод 1.2	—	—
	Перемотка	Вывод 1.3	—
Фаза 2	Ввод 1.1	Вывод 2.1	—
	Ввод 1.2	Перемотка	Вывод 2.2
	Перемотка	Ввод 1.3	Вывод 2.3
Фаза 3	Вывод 3.1	Ввод 2.1	Перемотка
	Вывод 3.2	Перемотка	Ввод 2.2
	Перемотка	Вывод 3.3	Ввод 2.3
Фаза 4	Ввод 3.1	Вывод 4.1	Перемотка
	Ввод 3.2	Перемотка	Вывод 4.2
	Перемотка	Ввод 3.3	Вывод 4.3

и т. д. Здесь "Вывод $k.j$" означает запись j-й трети k-го выводного файла, а "Ввод $k.j$" — ее чтение. В конце концов, будет исключено все время перемотки, если перемотка выполняется, по крайней мере, вдвое быстрее чтения/записи. Подобная процедура, в которой вывод в каждой фазе разделяется между лентами, называется расщеплением лент.

В работе R. L. McAllester, *CACM* **7** (1964), 158–159, показано, что расщепление лент приводит к эффективному методу совмещения времени перемотки в многофазном слиянии. Этот метод можно использовать с четырьмя или более лентами, и он осуществляет $(T-2)$-путевое слияние.

Предположим снова, что имеется шесть лент, и попытаемся построить схему слияния, которая работает следующим образом, расщепляя вывод на каждом уровне (буквы "I", "O" и "R" обозначают соответственно ввод, вывод и перемотку).

Уровень	T1	T2	T3	T4	T5	T6	Число выводимых серий
7	I	I	I	I	R	O	u_7
	I	I	I	I	O	R	v_7
6	I	I	I	R	O	I	u_6
	I	I	I	O	R	I	v_6
5	I	I	R	O	I	I	u_5
	I	I	O	R	I	I	v_5
4	I	R	O	I	I	I	u_4
	I	O	R	I	I	I	v_4
3	R	O	I	I	I	I	u_3
	O	R	I	I	I	I	v_3
2	O	I	I	I	I	R	u_2
	R	I	I	I	I	O	v_2
1	I	I	I	I	R	O	u_1
	I	I	I	I	O	R	v_1
0	I	I	I	R	O	I	u_0
	I	I	I	O	R	I	v_0 (21)

Чтобы по окончании работы получить одну серию на ленте T4, а остальные ленты сделать пустыми, необходимо иметь

$$\begin{aligned}
v_0 &= 1,\\
u_0 + v_1 &= 0,\\
u_1 + v_2 &= u_0 + v_0,\\
u_2 + v_3 &= u_1 + v_1 + u_0 + v_0,\\
u_3 + v_4 &= u_2 + v_2 + u_1 + v_1 + u_0 + v_0,\\
u_4 + v_5 &= u_3 + v_3 + u_2 + v_2 + u_1 + v_1 + u_0 + v_0,\\
u_5 + v_6 &= u_4 + v_4 + u_3 + v_3 + u_2 + v_2 + u_1 + v_1 + u_0 + v_0
\end{aligned}$$

и т. д. В общем случае требуется, чтобы

$$u_n + v_{n+1} = u_{n-1} + v_{n-1} + u_{n-2} + v_{n-2} + u_{n-3} + v_{n-3} + u_{n-4} + v_{n-4} \qquad (22)$$

при всех $n \geq 0$, если считать $u_j = v_j = 0$ при всех $j < 0$.

Эти уравнения не имеют единственного решения. В самом деле, если положить все u равными нулю, то получится обычное многофазное слияние, причем одна лента будет лишней! Но если выбрать $u_n \approx v_{n+1}$, то время перемотки будет удовлетворительно совмещено.

В указанной работе Мак-Аллестер предложил взять

$$u_n = v_{n-1} + v_{n-2} + v_{n-3} + v_{n-4},$$
$$v_{n+1} = u_{n-1} + u_{n-2} + u_{n-3} + u_{n-4},$$

так что последовательность

$$\langle x_0, x_1, x_2, x_3, x_4, x_5, \ldots \rangle = \langle v_0, u_0, v_1, u_1, v_2, u_2, \ldots \rangle$$

удовлетворяет однородному рекуррентному соотношению $x_n = x_{n-3} + x_{n-5} + x_{n-7} + x_{n-9}$. Оказалось, однако, что лучше положить

$$\begin{aligned} v_{n+1} &= u_{n-1} + v_{n-1} + u_{n-2} + v_{n-2}, \\ u_n &= u_{n-3} + v_{n-3} + u_{n-4} + v_{n-4}. \end{aligned} \tag{23}$$

Эта последовательность не только немного лучше по времени слияния; ее большое преимущество состоит в том, что соответствующее время слияния можно проанализировать математически. Вариант Мак-Аллестера для анализа крайне труден, потому что в одной фазе могут встречаться серии разной длины; мы увидим, что такого не может случиться, если справедливо (23).

Можно вывести число серий на каждой ленте на каждом уровне, двигаясь назад по схеме (21), и получить следующую схему сортировки.

Уровень	T1	T2	T3	T4	T5	T6	Время записи	Время перемотки
	1^{23}	1^{21}	1^{17}	1^{10}	—	1^{11}	82	23
7	1^{19}	1^{17}	1^{13}	1^{6}	R	$1^{11}4^{4}$	$4 \times 4 = 16$	$82 - 23$
	1^{13}	1^{11}	1^{7}	—	4^{6}	R	$6 \times 4 = 24$	27
6	1^{10}	1^{8}	1^{4}	R	4^{9}	$1^{8}4^{4}$	$3 \times 4 = 12$	10
	1^{6}	1^{4}	—	4^{4}	R	$1^{4}4^{4}$	$4 \times 4 = 16$	36
5	1^{5}	1^{3}	R	$4^{4}7^{1}$	4^{8}	$1^{3}4^{4}$	$1 \times 7 = 7$	17
	1^{2}	—	7^{3}	R	4^{5}	4^{4}	$3 \times 7 = 21$	23
4	1^{1}	R	$7^{3}13^{1}$	$4^{3}7^{1}$	4^{4}	4^{3}	$1 \times 13 = 13$	21
	—	13^{1}	R	$4^{2}7^{1}$	4^{3}	4^{2}	$1 \times 13 = 13$	34
3	R	$13^{1}19^{1}$	$7^{2}13^{1}$	$4^{1}7^{1}$	4^{2}	4^{1}	$1 \times 19 = 19$	23
	19^{1}	R	$7^{1}13^{1}$	7^{1}	4^{1}	—	$1 \times 19 = 19$	32
2	$19^{1}31^{0}$	$13^{1}19^{1}$	$7^{1}13^{1}$	7^{1}	4^{1}	R	$0 \times 31 = 0$	27
	R	19^{1}	13^{1}	7^{0}	—	31^{1}	$1 \times 31 = 31$	19
1	$19^{1}31^{0}$	19^{1}	13^{1}	7^{0}	R	$31^{1}52^{0}$	$0 \times 52 = 0$	$\max(36, 31, 23)$
	$19^{1}31^{0}$	19^{1}	13^{1}	—	52^{0}	R	$0 \times 52 = 0$	
0	$19^{1}31^{0}$	19^{1}	13^{1}	R	$52^{0}82^{0}$	$31^{1}52^{0}$	$0 \times 82 = 0$	
	(31^{0})	(19^{0})	—	82^{1}	(R)	(52^{0})	$1 \times 82 = 82$	0

Несовмещенная перемотка встречается только при перемотке вводной ленты T5 (82 единицы) в течение первой половины фазы второго уровня (27 единиц) и в течение окончательных фаз "фиктивного слияния" на уровнях 1 и 0 (36 единиц). Таким образом, время работы можно оценить величиной $273t + 145r$; для алгоритма D соответствующий параметр $268t + 208r$ почти всегда хуже.

Нетрудно видеть (см. упр. 23), что длины серий, выводимых во время каждой фазы, суть

$$4, 4, 7, 13, 19, 31, 52, 82, 133, \ldots, \tag{24}$$

при этом последовательность $\langle t_1, t_2, t_3, \ldots \rangle$ удовлетворяет закону

$$t_n = t_{n-2} + 2t_{n-3} + t_{n-4}, \tag{25}$$

если считать $t_n = 1$ при $n \le 0$. Можно также проанализировать оптимальное размещение фиктивных серий, рассмотрев цепочки чисел слияний, как для стандартного многофазного метода [ср. с (8)].

Уровень	T1	T2	T3	T4	T6	Окончательный результат
1	1	1	1	1	—	T5
2	1	1	1	—	1	T4
3	21	21	2	2	1	T3
4	2221	222	222	22	2	T2
5	23222	23222	2322	23	222	T1
6	333323222	33332322	333323	3333	2322	T6
. . .	. . .	. . .	. . .	. . .	. . .	. . .
n	A_n	B_n	C_n	D_n	E_n	T(k)
n+1	$(A''_nE_n+1)B_n$	$(A''_nE_n+1)C_n$	$(A''_nE_n+1)D_n$	A''_nE_n+1	A'_n	T(k−1)
. . .	. . .	. . .	. . .	. . .	. . .	. . .

(26)

Здесь $A_n = A'_nA''_n$ и A''_n состоит из последних u_n чисел слияний A_n. Приведенное выше правило перехода с уровня n на уровень $n+1$ справедливо для *любой* схемы, удовлетворяющей (22). Если определить u_n и v_n, как в (23), то цепочки $A_n, \ldots, E_n$ можно выразить в следующем, довольно простом, виде [ср. с (9)]:

$$\begin{aligned} A_n &= (W_{n-1}W_{n-2}W_{n-3}W_{n-4}) + 1, \\ B_n &= (W_{n-1}W_{n-2}W_{n-3}) + 1, \\ C_n &= (W_{n-1}W_{n-2}) + 1, \\ D_n &= (W_{n-1}) + 1, \\ E_n &= (W_{n-2}W_{n-3}) + 1, \end{aligned} \tag{27}$$

где

$$\begin{aligned} &W_n = (W_{n-3}W_{n-4}W_{n-2}W_{n-3}) + 1 \qquad \text{при } n > 0, \\ &W_0 = 0 \qquad \text{и} \qquad W_n = \epsilon \quad \text{при } n < 0. \end{aligned} \tag{28}$$

Исходя из этих соотношений, легко подробно проанализировать случай для шести лент.

В общем случае, если имеется $T \ge 5$ лент, положим $P = T - 2$ и определим последовательности $\langle u_n \rangle$, $\langle v_n \rangle$ по правилам

$$\begin{aligned} v_{n+1} &= u_{n-1} + v_{n-1} + \cdots + u_{n-r} + v_{n-r}, \\ u_n &= u_{n-r-1} + v_{n-r-1} + \cdots + u_{n-P} + v_{n-P} \qquad \text{при } n \ge 0, \end{aligned} \tag{29}$$

где $r = \lfloor P/2 \rfloor$, $v_0 = 1$ и $u_n = v_n = 0$ при $n < 0$. Если $w_n = u_n + v_n$, то имеем

$$w_n = w_{n-2} + \cdots + w_{n-r} + 2w_{n-r-1} + w_{n-r-2} + \cdots + w_{n-P} \quad \text{при } n > 0; \tag{30}$$

$w_0 = 1$ и $w_n = 0$ при $n < 0$. При начальном распределении для уровня $n+1$ на ленту k помещается $w_n + w_{n-1} + \cdots + w_{n-P+k}$ серий при $1 \le k \le P$ и $w_{n-1} + \cdots + w_{n-r}$ — на ленту T; лента $T-1$ используется для ввода. Затем u_n серий сливаются на ленту T, в то время как лента $T-1$ перематывается; v_n серий сливаются на $T-1$, пока T перематывается; u_{n-1} серий сливаются на $T-1$, пока $T-2$ перематывается, и т. д.

Таблица 6

ПРИБЛИЗИТЕЛЬНЫЕ ДАННЫЕ О ХОДЕ СОРТИРОВКИ МЕТОДОМ МНОГОФАЗНОГО СЛИЯНИЯ С РАСЩЕПЛЕНИЕМ ЛЕНТ

Ленты	Фазы	Проходы	Проходы/ фазы, %	Отношение роста
4	$2.885 \ln S + 0.000$	$1.443 \ln S + 1.000$	50	1.4142136
5	$2.078 \ln S + 0.232$	$0.929 \ln S + 1.022$	45	1.6180340
6	$2.078 \ln S - 0.170$	$0.752 \ln S + 1.024$	36	1.6180340
7	$1.958 \ln S - 0.408$	$0.670 \ln S + 1.007$	34	1.6663019
8	$2.008 \ln S - 0.762$	$0.624 \ln S + 0.994$	31	1.6454116
9	$1.972 \ln S - 0.987$	$0.595 \ln S + 0.967$	30	1.6604077
10	$2.013 \ln S - 1.300$	$0.580 \ln S + 0.941$	29	1.6433803
20	$2.069 \ln S - 3.164$	$0.566 \ln S + 0.536$	27	1.6214947

В табл. 6 показаны приблизительные данные о ходе выполнения этой процедуры, когда S не слишком мало. В столбце "Проходы/фазы" примерно показано, какая часть всего файла перематывается во время каждой половины фазы и какая часть файла записывается за время каждой полной фазы. *Метод расщепления лент превосходит стандартный многофазный метод на шести или более лентах* и, вероятно, также на пяти лентах, по крайней мере для больших S.

Если $T = 4$, то указанная процедура стала бы, по существу, эквивалентной сбалансированному двухпутевому слиянию *без* совмещения времени перемотки, так как w_{2n+1} было бы равно 0 при всех n. Поэтому элементы табл. 6 при $T = 4$ были получены посредством небольшой модификации, состоящей в том, что полагалось $v_2 = 0, u_1 = 1, v_1 = 0, u_0 = 0, v_0 = 1$ и $v_{n+1} = u_{n-1} + v_{n-1}$, $u_n = u_{n-2} + v_{n-2}$ при $n \geq 2$. Это позволяет построить очень интересную схему сортировки (см. упр. 25 и 26).

УПРАЖНЕНИЯ

1. [*16*] На рис. 69 указан порядок, в котором алгоритм D распределяет по пяти лентам серии 34–65. В каком порядке распределяются серии 1–33?

▶ **2.** [*21*] Верно ли, что после двух фаз слияния в алгоритме D, т. е. когда мы во второй раз достигаем шага D6, все фиктивные серии исчезают?

▶ **3.** [*22*] Докажите, что по окончании шага D4 всегда выполняется условие `D[1]` $\geq$ `D[2]` $\geq \cdots \geq$ `D[`T`]`. Объясните важность этого условия для правильной работы на шагах D2 и D3.

4. [*M20*] Выведите производящие функции (7).

5. [*HM26*] (Э. П. Майлс (E. P. Miles, Jr.), 1960.) Докажите, что при всех $p \geq 2$ многочлен $f_p(z) = z^p - z^{p-1} - \cdots - z - 1$ имеет p различных корней, из которых ровно один превосходит 1 по абсолютной величине. [*Указание.* Рассмотрите многочлен $z^{p+1} - 2z^p + 1$.]

6. [*HM24*] Цель этого упражнения — рассмотреть способ составления табл. 1, 5 и 6. Предположим, что имеется схема слияния, свойства которой следующим образом характеризуются многочленами $p(z)$ и $q(z)$. (i) Число начальных серий в "точном распределении", требующем n фаз слияния, равно $[z^n]\,p(z)/q(z)$. (ii) Число начальных серий, обрабатываемых в течение этих n фаз слияния, равно $[z^n]\,p(z)/q(z)^2$. (iii) У многочлена $q(z^{-1})$ есть "главный корень" α, такой, что $q(\alpha^{-1}) = 0$, $q'(\alpha^{-1}) \neq 0$, $p(\alpha^{-1}) \neq 0$, и из $q(\beta^{-1}) = 0$ следует, что $\beta = \alpha$ или $|\beta| < |\alpha|$.

Докажите, что существует $\epsilon > 0$, такое, что если S равно числу серий при точном распределении, требующем n фаз слияния, а во время выполнения этих фаз обрабатывается

ρS серий, то $n = a\ln S + b + O(S^{-\epsilon})$ и $\rho = c\ln S + d + O(S^{-\epsilon})$, где

$$a = (\ln\alpha)^{-1}, \qquad b = -a\ln\left(\frac{p(\alpha^{-1})}{-q'(\alpha^{-1})}\right) - 1, \qquad c = a\,\frac{\alpha}{-q'(\alpha^{-1})},$$
$$d = \frac{(b+1)\alpha - p'(\alpha^{-1})/p(\alpha^{-1}) + q''(\alpha^{-1})/q'(\alpha^{-1})}{-q'(\alpha^{-1})}.$$

7. [*HM22*] Пусть α_p — главный корень многочлена $f_p(z)$ из упр. 5. Каково асимптотическое поведение α_p при $p \to \infty$?

8. [*M20*] (Э. Нетто (E. Netto), 1901.) Пусть $N_m^{(p)}$ есть число способов выражения m в виде упорядоченной суммы целых чисел $\{1, 2, \ldots, p\}$. Например, если $p = 3$ и $m = 5$, то имеется 13 способов: $1+1+1+1+1 = 1+1+1+2 = 1+1+2+1 = 1+1+3 = 1+2+1+1 = 1+2+2 = 1+3+1 = 2+1+1+1 = 2+1+2 = 2+2+1 = 2+3 = 3+1+1 = 3+2$. Покажите, что $N_m^{(p)}$ являются обобщенными числами Фибоначчи.

9. [*M20*] Пусть $K_m^{(p)}$ — число последовательностей из нулей и единиц, таких, что в них нет p последовательных единиц. Например, если $p = 3$ и $m = 5$, имеется 24 варианта: 00000, 00001, 00010, 00011, 00100, 00101, 00110, 01000, 01001, ..., 11011. Покажите, что $K_m^{(p)}$ являются обобщенными числами Фибоначчи.

10. [*M27*] (*Система счисления с обобщенными числами Фибоначчи.*) Докажите, что каждое неотрицательное целое n имеет единственное представление в виде суммы различных чисел Фибоначчи p-го порядка $F_j^{(p)}$ при $j \geq p$, удовлетворяющее условию, что не используются никакие p-последовательные числа Фибоначчи.

11. [*M24*] Докажите, что n-й элемент цепочки Q_∞ в (12) равен количеству различных чисел Фибоначчи в представлении элемента $n-1$ числами Фибоначчи пятого порядка (см. упр. 10).

▶ **12.** [*M18*] Найдите зависимость между степенями матрицы $\begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{pmatrix}$ и точным фибоначчиевым распределением в (1).

▶ **13.** [*22*] Докажите следующее интересное свойство точных фибоначчиевых распределений: если окончательный вывод оказывается на ленте номер T, то число серий на всех других лентах *нечетное*; если окончательный вывод оказывается на некоторой ленте, отличной от T, то число серий будет *нечетным* на этой ленте и *четным* на остальных (см. (1)).

14. [*M35*] Пусть $T_n(x) = \sum_{k\geq 0} T_{nk}x^k$, где $T_n(x)$ — многочлены, определенные в (16).

a) Покажите, что для каждого k существует число $n(k)$, такое, что $T_{1k} \leq T_{2k} \leq \cdots \leq T_{n(k)k} > T_{(n(k)+1),k} \geq \cdots$.

b) При условии, что $T_{n'k'} < T_{nk'}$ и $n' < n$, докажите, что $T_{n'k} \leq T_{nk}$ для всех $k \geq k'$.

c) Докажите, что существует неубывающая последовательность $\langle M_n \rangle$, такая, что $\Sigma_n(S) = \min_{j\geq 1}\Sigma_j(S)$ при $M_n \leq S < M_{n+1}$, но $\Sigma_n(S) > \min_{j\geq 1}\Sigma_j(S)$ при $S \geq M_{n+1}$ (см. (19)).

15. [*M43*] Верно ли, что условие $\Sigma_{n-1}(m) < \Sigma_n(m)$ влечет за собой $\Sigma_n(m) \leq \Sigma_{n+1}(m) \leq \Sigma_{n+2}(m) \leq \cdots$. (Такой результат сильно упростил бы вычисление табл. 2.)

16. [*HM43*] Определите асимптотическое поведение многофазного слияния с оптимальным распределением фиктивных серий.

17. [*32*] Верно ли, что серии для оптимального многофазного распределения можно разместить таким образом, что распределение $S+1$ начальных серий получится путем добавления одной серии (на соответствующую ленту) к распределению S начальных серий?

18. [*30*] Верно ли, что оптимальное многофазное распределение дает наилучшую возможную схему слияния в том смысле, что суммарное количество обрабатываемых начальных серий минимально, если требуется, чтобы начальные серии размещались не более чем на $T-1$ лентах? (Временем перемотки пренебречь.)

19. [*21*] Составьте таблицу, аналогичную (1), для многофазного метода сортировки Карона для шести лент.

20. [*M24*] Какая производящая функция для кароновской многофазной сортировки на шести лентах соответствует (7) и (16)? Какие соотношения, аналогичные (9) и (27), определяют строки чисел слияний?

21. [*11*] Что должно появиться на уровне 7 в (26)?

22. [*M21*] Каждый член последовательности (24) приблизительно равен сумме двух предыдущих членов. Наблюдается ли это явление для остальных членов последовательности? Сформулируйте и докажите теорему о $t_n - t_{n-1} - t_{n-2}$.

▶ **23.** [*29*] Какие изменения необходимо выполнить в (25), (27) и (28), если (23) заменить на $v_{n+1} = u_{n-1} + v_{n-1} + u_{n-2}$, $u_n = v_{n-2} + u_{n-3} + v_{n-3} + u_{n-4} + v_{n-4}$?

24. [*HM41*] Вычислите асимптотическое поведение многофазной процедуры с расщеплением лент, если элемент v_{n+1} определен как сумма первых q членов $u_{n-1} + v_{n-1} + \cdots + u_{n-P} + v_{n-P}$ при различных $P = T-2$ и $0 \le q \le 2P$. (В тексте раздела рассматривается только случай, когда $q = 2\lfloor P/2 \rfloor$; см. упр. 23.)

25. [*19*] Продемонстрируйте, как многофазное слияние с расщеплением лент, упомянутое в конце этого раздела, сортировало бы 32 начальные серии. (Проанализируйте каждую фазу, как в тексте, в примере с 82 сериями на шести лентах.)

26. [*M21*] Проанализируйте ход выполнения многофазного слияния с расщеплением лент на четырех лентах при $S = 2^n$ и при $S = 2^n + 2^{n-1}$ (см. упр. 25).

27. [*23*] Если начальные серии распределены на лентах в соответствии с точным распределением, то многофазная стратегия превращается просто в "слияние до опустошения". Сначала сливаем серии со всех непустых входных лент, пока одна из них не станет пустой; затем используем эту ленту как следующую выводную, а предыдущую выводную ленту — как вводную.

Верно ли, что стратегия "сливать до опустошения" всегда позволяет выполнять сортировку независимо от того, как распределены начальные серии, при условии, что мы распределяем их, по крайней мере, на две ленты? (Одна лента, конечно, будет оставлена пустой, чтобы она могла служить первой выводной лентой.)

28. [*M26*] В предыдущем упражнении определено весьма большое семейство схем слияния. Покажите, что многофазная схема — *наилучшая* из них в следующем смысле: если имеется шесть лент и мы рассматриваем класс всех начальных распределений (a, b, c, d, e), таких, что стратегия "сливать до опустошения" требует n или меньше фаз для сортировки, то $a+b+c+d+e \le t_n$, где t_n — соответствующее число для многофазной сортировки (1).

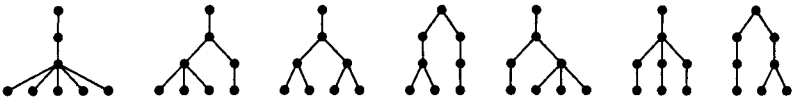
29. [*M47*] В упр. 28 показано, что многофазное распределение оптимально среди всех схем "сливать до опустошения" в смысле минимальности числа фаз. Но является ли оно оптимальным также в смысле минимальности числа проходов?

Пусть числа a и b взаимно простые, и предположим, что $a+b$ есть число Фибоначчи F_n. Верно ли следующее предположение, высказанное Р. М. Карпом (R. M. Karp): число начальных серий, которые обрабатываются схемой "сливать до опустошения", начинающейся

с распределения (a, b), больше или равно $((n-5)F_{n+1}+(2n+2)F_n)/5$? (Указанное значение достигается, когда $a = F_{n-1}$, $b = F_{n-2}$.)

30. [*42*] Составьте таблицу, аналогичную табл. 2, для многофазного слияния с расщеплением лент.

31. [*M22*] (Р. Кемп (R. Kemp).) Пусть $K_d(n)$ — число n-узловых упорядоченных деревьев, в которых каждый лист находится на расстоянии d от корня. Например, в изображенных ниже деревьях $K_3(8) = 7$.



Покажите, что $K_d(n)$ является обобщенным числом Фибоначчи, и найдите однозначное соответствие между такими деревьями и упорядоченным разбиением, которое рассматривалось в упр. 8.

*5.4.3. Каскадное слияние

Другая основная схема, называемая каскадным слиянием, на самом деле была изобретена раньше многофазной [см. B. K. Betz, W. C. Carter, *ACM National Conf.* **14** (1959), Paper 14]. Ниже, в таблице, этот подход иллюстрируется для шести лент и 190 начальных серий с использованием обозначений из раздела 5.4.2.

	T1	T2	T3	T4	T5	T6	Обработанные начальные серии
Проход 1	1^{55}	1^{50}	1^{41}	1^{29}	1^{15}	—	190
Проход 2	—	$*1^{5}$	2^{9}	3^{12}	4^{14}	5^{15}	190
Проход 3	15^{5}	14^{4}	12^{3}	9^{2}	$*5^{1}$	—	190
Проход 4	—	$*15^{1}$	29^{1}	41^{1}	50^{1}	55^{1}	190
Проход 5	190^{1}	—	—	—	—	—	190

Каскадное слияние подобно многофазному начинается с точного распределения серий по лентам, хотя правила точного распределения отличны от правил из раздела 5.4.2. Каждая строка таблицы представляет полный проход по *всем* данным. Проход 2, например, получается посредством выполнения пятипутевого слияния с $\{T1, T2, T3, T4, T5\}$ на T6, пока T5 не станет пустой (при этом на T6 помещаются 15 серий относительной длиной 5), затем четырехпутевого слияния с $\{T1, T2, T3, T4\}$ на T5, затем трехпутевого слияния на T4, двухпутевого слияния на T3 и, наконец, однопутевого слияния (операции копирования) с T1 на T2. Проход 3 получается таким же образом, путем выполнения сначала пятипутевого слияния, пока одна лента не станет пустой, затем — четырехпутевого и т. д. (Похоже, что разделу книги следовало бы присвоить номер 5.4.3.2.1, а не 5.4.3!)

Ясно, что операции копирования излишни и их можно было бы опустить. Однако фактически в случае для шести лент это копирование отнимает только небольшую часть всего времени. Элементы, которые получаются в результате простого копирования, отмечены в приведенной выше таблице звездочкой. Только 25 из 950 обрабатываемых серий принадлежат этому классу. Бóльшая часть времени уходит на пятипутевое и четырехпутевое слияния.

На первый взгляд может показаться, что каскадная схема проигрывает в сравнении с многофазной, так как стандартная многофазная схема всегда использует

Таблица 1

ПРИБЛИЗИТЕЛЬНЫЕ ДАННЫЕ О ХОДЕ СОРТИРОВКИ МЕТОДОМ КАСКАДНОГО СЛИЯНИЯ

Ленты	Проходы (с копированием)	Проходы (без копирования)	Отношение роста
3	$2.078 \ln S + 0.672$	$1.504 \ln S + 0.992$	1.6180340
4	$1.235 \ln S + 0.754$	$1.102 \ln S + 0.820$	2.2469796
5	$0.946 \ln S + 0.796$	$0.897 \ln S + 0.800$	2.8793852
6	$0.796 \ln S + 0.821$	$0.773 \ln S + 0.808$	3.5133371
7	$0.703 \ln S + 0.839$	$0.691 \ln S + 0.822$	4.1481149
8	$0.639 \ln S + 0.852$	$0.632 \ln S + 0.834$	4.7833861
9	$0.592 \ln S + 0.861$	$0.587 \ln S + 0.845$	5.4189757
10	$0.555 \ln S + 0.869$	$0.552 \ln S + 0.854$	6.0547828
20	$0.397 \ln S + 0.905$	$0.397 \ln S + 0.901$	12.4174426

$(T-1)$-путевое слияние в то время как каскадная использует $(T-1)$-путевое, $(T-2)$-путевое, $(T-3)$-путевое и т. д. В действительности же для шести и более лент каскадная схема асимптотически *лучше*, чем многофазная. Как мы видели в разделе 5.4.2, высокий порядок слияния не является гарантией эффективности. В табл. 1 показаны характеристики выполнения каскадного слияния по аналогии с подобной таблицей из раздела 5.4.2.

Нетрудно, рассуждая "в обратном направлении" от конечного состояния $(1, 0, \ldots, 0)$, вывести "точные распределения" для каскадного слияния. В случае для шести лент имеем следующее.

Уровень	T1	T2	T3	T4	T5
0	1	0	0	0	0
1	1	1	1	1	1
2	5	4	3	2	1
3	15	14	12	9	5
4	55	50	41	29	15
5	190	175	146	105	55
. . .	. . .	. . .	. . .	. . .	. . .
n	a_n	b_n	c_n	d_n	e_n
$n+1$	$a_n+b_n+c_n+d_n+e_n$	$a_n+b_n+c_n+d_n$	$a_n+b_n+c_n$	a_n+b_n	a_n

(1)

Отметим интересное свойство этих чисел — их относительные величины являются также и длинами диагоналей правильного $(2T-1)$-угольника. Например, пять диагоналей одиннадцатиугольника на рис. 73 имеют относительные длины, очень близкие к 190, 175, 146, 105 и 55! Мы докажем ниже в этом разделе такой замечательный факт и увидим, что значения относительного времени, затрачиваемого на $(T-1)$-, $(T-2)$-, ..., 1-путевое слияние, приблизительно пропорциональны *квадратам* длин этих диагоналей.

Начальное распределение серий. Если число начальных серий в действительности не есть число Фибоначчи, можно, как обычно, вставить фиктивные серии. Даже из поверхностного анализа ситуации ясно, что метод приписывания фиктивных серий здесь работать не будет, так как при каскадном слиянии всегда выполняются

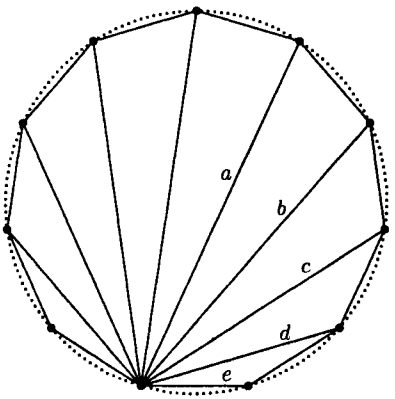


Рис. 73. Геометрическая интерпретация каскадных чисел.

полные проходы. Если имеется 190 начальных серий, то каждая запись обрабатывается пять раз, как в приведенном выше примере, но если имеется 191 серия, то, очевидно, следует увеличить уровень. Теперь каждая запись будет обрабатываться шесть раз. К счастью, на практике можно избежать такого резкого скачка. Дэвид Э. Фергюсон (David E. Ferguson) сумел так распределить начальные серии, что многие операции во время первой фазы слияния свелись к копированию содержимого ленты. Обойдя такие копирования (просто изменив "логические" номера ленточных устройств по отношению к "физическим" номерам, как в алгоритме 5.4.2D), получим относительно плавный переход от одного уровня на другой, как изображено на рис. 74.

Предположим, что (a, b, c, d, e), где $a \geq b \geq c \geq d \geq e$ — точное распределение. Изменив соответствие между логическими и физическими ленточными устройствами, можно представить, что реальное распределение — (e, d, c, b, a), т. е. a серий на T5, b — на T4 и т. д. Следующее точное распределение — это $(a+b+c+d+e$, $a+b+c+d$, $a+b+c$, $a+b$, $a)$; если же ввод исчерпывается прежде, чем мы достигаем этого следующего уровня, будем считать, что на лентах содержится соответственно $(D_1, D_2, D_3, D_4, D_5)$ фиктивных серий, где

$$D_1 \leq a+b+c+d, \quad D_2 \leq a+b+c, \quad D_3 \leq a+b, \quad D_4 \leq a, \quad D_5 = 0;$$
$$D_1 \geq D_2 \geq D_3 \geq D_4 \geq D_5. \tag{2}$$

Мы вольны представлять себе, что эти фиктивные серии появляются на лентах в любом удобном месте. Предполагается, что первый проход слияния сначала даст a серий посредством пятипутевого слияния, затем — b серий посредством четырехпутевого и т. д. Наша цель состоит в размещении фиктивных серий таким образом, чтобы заменить слияние копированием. Удобно выполнять первый проход слияния следующим образом.

1. Если $D_4 = a$, то вычесть a из всех D_1, D_2, D_3, D_4 и заявить, что T5 — результат слияния. Если $D_4 < a$, то произвести слияние a серий с лент T1 по T5, используя минимально возможное число фиктивных серий на лентах от T1 до T5

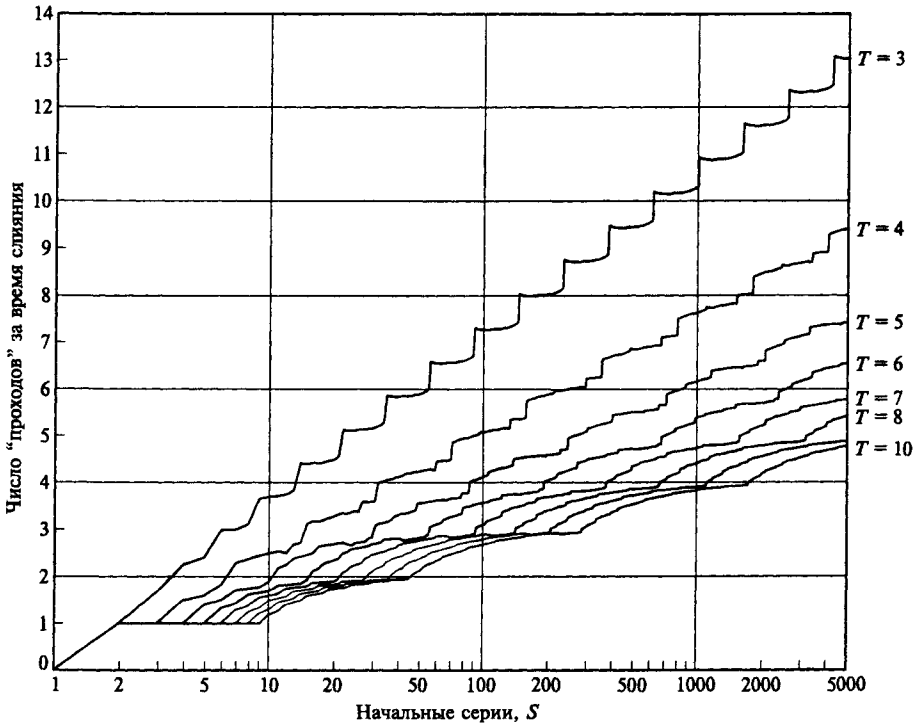


Рис. 74. Эффективность каскадного слияния с распределением по алгоритму D.

так, чтобы новые значения D_1, D_2, D_3, D_4 удовлетворяли соотношениям

$$D_1 \le b + c + d, \quad D_2 \le b + c, \quad D_3 \le b, \quad D_4 = 0; \quad D_1 \ge D_2 \ge D_3 \ge D_4. \qquad (3)$$

Таким образом, если D_2 было первоначально $\le b + c$, ни одна фиктивная серия с этой ленты на данном шаге не используется. В то же время, если $b+c < D_2 \le a+b+c$, используется ровно $D_2 - b - c$ фиктивных серий.

2. (Этот шаг аналогичен шагу 1, но с некоторым "сдвигом".) Если $D_3 = b$, то вычесть b из всех D_1, D_2, D_3 и объявить, что T4 — результат слияния. Если $D_3 < b$, то слить b серий с лент T1–T4, уменьшая, если необходимо, число фиктивных серий, чтобы достичь

$$D_1 \le b + c, \quad D_2 \le b, \quad D_3 = 0; \qquad D_1 \ge D_2 \ge D_3.$$

3. И так далее.

Метод распределения серий по лентам Фергюсона можно проиллюстрировать, рассмотрев переход с уровня 3 на уровень 4 в (1). Допустим, что на "логических" лентах (T1, ..., T5) содержалось соответственно (5, 9, 12, 14, 15) серий и что необходимо довести это количество до (55, 50, 41, 29, 15). Данная процедура кратко описана в табл. 2. Сначала помещаем девять серий на T1, затем (3, 12) — на T1 и T2 и т. д. Если ввод исчерпывается, скажем, на шаге (3, 2), то "сэкономленная величина" составляет 15 + 9 + 5. Это означает, что мы избавляемся от пятипутевого слияния

Таблица 2

ПРИМЕР ПОШАГОВОГО РАСПРЕДЕЛЕНИЯ

	Добавить к T1	Добавить к T2	Добавить к T3	Добавить к T4	Добавить к T5	"Сэкономлено"
Шаг (1,1)	9	0	0	0	0	15+14+12+5
Шаг (2,2)	3	12	0	0	0	15+14+9+5
Шаг (2,1)	9	0	0	0	0	15+14+5
Шаг (3,3)	2	2	14	0	0	15+12+5
Шаг (3,2)	3	12	0	0	0	15+9+5
Шаг (3,1)	9	0	0	0	0	15+5
Шаг (4,4)	1	1	1	15	0	14+5
Шаг (4,3)	2	2	14	0	0	12+5
Шаг (4,2)	3	12	0	0	0	9+5
Шаг (4,1)	9	0	0	0	0	5

15 серий, двухпутевого слияния 9 серий и однопутевого слияния 5 серий посредством присвоения фиктивных серий. Другими словами, $15+9+5$ серий, присутствующих на уровне 3, не обрабатываются в течение первой фазы слияния.

Следующий алгоритм детально описывает этот процесс.

Алгоритм C (*Сортировка методом каскадного слияния со специальным распределением*). Данный алгоритм (рис. 75) распределяет начальные серии по лентам серия за серией, пока запас начальных серий не исчерпается. Затем он определяет, как следует выполнять слияние лент, предполагая, что имеется $T \geq 3$ накопителей на магнитных лентах, при этом используется самое большое $(T-1)$-путевое слияние и ненужные однопутевые слияния устраняются. Лента T может использоваться для хранения исходных данных, так как на нее не попадает ни одна начальная серия. Алгоритм работает со следующими массивами.

`A[j]`, $1 \leq j \leq T$: Последнее точное каскадное распределение, которое было достигнуто

`AA[j]`, $1 \leq j \leq T$: Точное каскадное распределение, к которому мы стремимся

`D[j]`, $1 \leq j \leq T$: Число фиктивных серий, которые, как мы предполагаем, присутствуют на логическом накопителе на магнитной ленте с номером j

`M[j]`, $1 \leq j < T$: Максимальное число фиктивных серий, которые желательно иметь на логическом накопителе на магнитной ленте с номером j

`TAPE[j]`, $1 \leq j \leq T$: Номер физического накопителя на магнитной ленте, соответствующий логическому накопителю на магнитной ленте с номером j

C1. [Начальная установка.] Установить `A[k]` $\leftarrow$ `AA[k]` $\leftarrow$ `D[k]` $\leftarrow 0$ при $2 \leq k \leq T$. Установить `A[1]` $\leftarrow 0$, `AA[1]` $\leftarrow 1$, `D[1]` $\leftarrow 1$. Установить `TAPE[k]` $\leftarrow k$ при $1 \leq k \leq T$. Наконец, установить $i \leftarrow T-2$, $j \leftarrow 1$, $k \leftarrow 1$, $l \leftarrow 0$, $m \leftarrow 1$ и перейти к шагу C5. (Эти действия являются одним из способов начать работу непосредственно во внутреннем цикле, уже имея соответствующую установку управляющих переменных.)

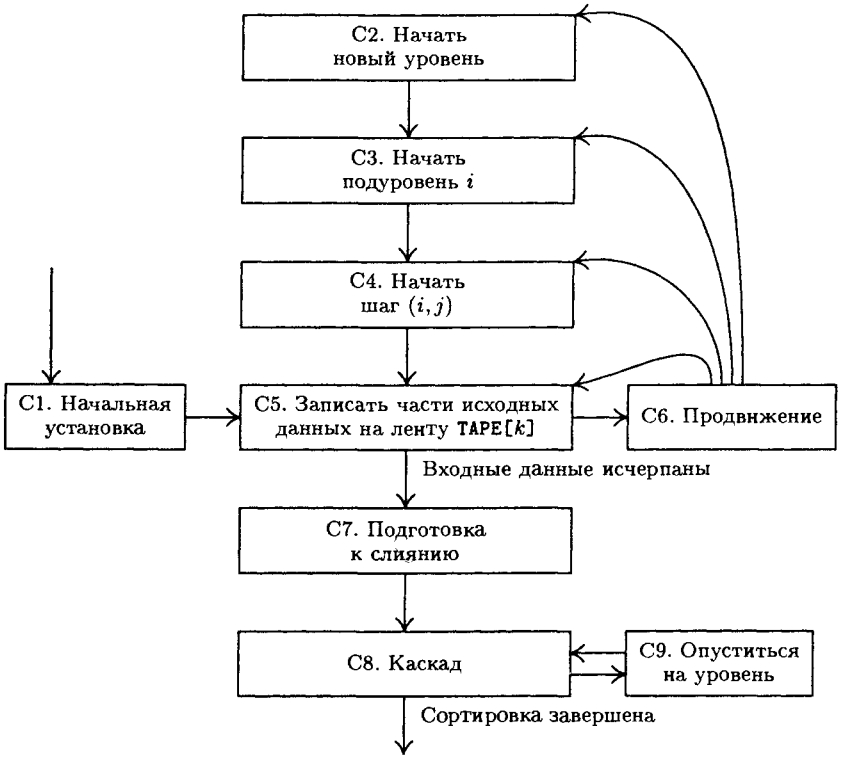


Рис. 75. Каскадное слияние со специальным распределением.

C2. [Начать новый уровень.] (Точное распределение только что получено. Но так как еще имеются исходные данные, необходимо подготовиться к следующему уровню.) Увеличить l на 1. Установить A[k] $\leftarrow$ AA[k] при $1 \le k \le T$ и AA[$T-k$] $\leftarrow$ AA[$T-k+1$] + A[k] при $k = 1, 2, \ldots, T-1$ (именно в таком порядке). Установить (TAPE[1], ..., TAPE[$T-1$]) $\leftarrow$ (TAPE[$T-1$], ..., TAPE[1]) и D[k] $\leftarrow$ AA[$k+1$] при $1 \le k < T$. Наконец, установить $i \leftarrow 1$.

C3. [Начать подуровень i.] Установить $j \leftarrow i$. (Переменные i и j представляют "шаг (i, j)" в табл. 2, иллюстрирующей метод Фергюсона.)

C4. [Начать шаг (i, j).] Установить $k \leftarrow j$ и $m \leftarrow$ A[$T-j-1$]. Если $m = 0$ и $i = j$, то установить $i \leftarrow T-2$ и вернуться к шагу C3; если $m = 0$ и $i \ne j$, вернуться к шагу C2. (Переменная m представляет собой число серий, которые должны быть записаны на ленту TAPE[k]; $m = 0$ бывает равно 0 только в случае, если $l = 1$.)

C5. [Записать части исходных данных на ленту TAPE[k].] Записать одну серию на ленту номер TAPE[k] и уменьшить D[k] на 1. Затем, если ввод исчерпан, перемотать все ленты и перейти к шагу C7.

C6. [Продвижение.] Уменьшить m на 1. Если $m > 0$, вернуться к шагу C5. В противном случае уменьшить k на 1; если $k > 0$, установить $m \leftarrow$

A[$T-j-1$] $-$ A[$T-j$] и вернуться к шагу C5, если $m > 0$. В противном случае уменьшить j на 1; если $j > 0$, перейти к шагу C4, в противном случае увеличить i на 1; если $i < T-1$, вернуться к шагу C3. В противном случае перейти к шагу C2.

C7. [Подготовка к слиянию.] (К этому моменту начальное распределение завершено и таблицы AA, D и TAPE описывают состояние всех лент в данный момент.) Установить M[k] $\leftarrow$ AA[$k+1$] при $1 \le k < T$ и установить FIRST $\leftarrow$ 1. (Переменная FIRST принимает ненулевое значение только во время первого прохода слияния.)

C8. [Каскад.] Если $l = 0$, остановиться; сортировка завершена, вывод находится на TAPE[1]. В противном случае при $p = T-1, T-2, \ldots, 1$ (именно в таком порядке) выполнять p-путевое слияние с лент TAPE[1], ..., TAPE[p] на TAPE[$p+1$] следующим образом.
Если $p = 1$, моделировать однопутевое слияние посредством обычной перемотки TAPE[2] и замены TAPE[1] $\leftrightarrow$ TAPE[2].
В противном случае, если FIRST $=$ 1 и D[$p-1$] $=$ M[$p-1$], моделировать p-путевое слияние, просто поменяв TAPE[p] $\leftrightarrow$ TAPE[$p+1$], перемотав TAPE[p] и выполнив вычитание M[$p-1$] из каждого D[1], ..., D[$p-1$], M[1], ..., M[$p-1$].
В противном случае вычесть M[$p-1$] из всех M[1], ..., M[$p-1$]. Затем слить по одной серии с каждой TAPE[j], такой, что $1 \le j \le p$ и D[j] $\le$ M[j]; вычесть единицу из каждого D[j], такого, что $1 \le j \le p$ и D[j] $>$ M[j], и помесить выводную серию на TAPE[$p+1$]. Продолжать работу, пока TAPE[p] не станет пустой. Затем перемотать TAPE[p] и TAPE[$p+1$].

C9. [Опуститься на уровень.] Уменьшить l на 1, установить FIRST $\leftarrow$ 0 и установить (TAPE[1], ..., TAPE[T]) $\leftarrow$ (TAPE[T], ..., TAPE[1]). (К этому моменту все D и M — нули; таковыми они и останутся.) Вернуться к шагу C8. ▌

На шагах C1–C6 алгоритма C выполняется распределение, на шагах C7–C9 — слияние. Эти две части совершенно независимы одна от другой, и можно было бы хранить M[k] и AA[$k+1$] в одних и тех же ячейках памяти.

Анализ каскадного слияния. Каскадное слияние поддается анализу с бóльшим трудом, чем многофазное. Но этот анализ особенно интересен, поскольку содержит много замечательных формул. Настоятельно рекомендуем читателям, интересующимся дискретной математикой, самостоятельно проанализировать каскадное распределение *прежде, чем читать дальше*. Ведь числа имеют так много необычных свойств, открывать которые для себя — одно удовольствие! Мы обсудим здесь лишь один из многих подходов, обращая особое внимание на методы получения результатов.

Для удобства рассмотрим случай для шести лент. При этом будем стараться получить формулы, которые обобщаются для любого T. Соотношения (1) позволяют получить первую основную систему:

$$
\begin{aligned}
a_n &= a_n && = \tbinom{0}{0}a_n,\\
b_n &= a_n - e_{n-1}\\
&= a_n - a_{n-2} && = \tbinom{1}{0}a_n - \tbinom{2}{2}a_{n-2},\\
c_n &= b_n - d_{n-1}\\
&= b_n - a_{n-2} - b_{n-2} && = \tbinom{2}{0}a_n - \tbinom{3}{2}a_{n-2} + \tbinom{4}{4}a_{n-4},\\
d_n &= c_n - e_{n-1}\\
&= c_n - a_{n-2} - b_{n-2} - c_{n-2} && = \tbinom{3}{0}a_n - \tbinom{4}{2}a_{n-2} + \tbinom{5}{4}a_{n-4} - \tbinom{6}{6}a_{n-6},\\
e_n &= d_n - b_{n-1}\\
&= d_n - a_{n-2} - b_{n-2} - c_{n-2} - d_{n-2} && = \tbinom{4}{0}a_n - \tbinom{5}{2}a_{n-2} + \tbinom{6}{4}a_{n-4} - \tbinom{7}{6}a_{n-6} + \tbinom{8}{8}a_{n-8}.
\end{aligned}
\tag{4}
$$

Обозначим $A(z) = \sum_{n\ge 0} a_n z^n, \ldots, E(z) = \sum_{n\ge 0} e_n z^n$ и определим многочлены

$$
\begin{aligned}
q_m(z) &= \binom{m}{0} - \binom{m+1}{2} z^2 + \binom{m+2}{4} z^4 - \cdots\\
&= \sum_k \binom{m+k}{2k} (-1)^k z^{2k} = \sum_{k=0}^{m} \binom{2m-k}{k} (-1)^{m-k} z^{2m-2k}.
\end{aligned}
\tag{5}
$$

Результат (4) кратко можно истолковать так, что $B(z) - q_1(z)A(z)$, $C(z) - q_2(z)A(z)$, $D(z) - q_3(z)A(z)$ и $E(z) - q_4(z)A(z)$ сводятся к конечным суммам, соответствующим граничным условиям, а именно — значениям $a_{-1}, a_{-2}, a_{-3}, \ldots$, которые появляются в (4) (при небольших n), но не в $A(z)$. Чтобы получить подходящие граничные условия, применим рекуррентное соотношение в обратную сторону для отрицательных уровней вплоть до уровня -8.

n	a_n	b_n	c_n	d_n	e_n
0	1	0	0	0	0
−1	0	0	0	0	1
−2	1	−1	0	0	0
−3	0	0	0	−1	2
−4	2	−3	1	0	0
−5	0	0	1	−4	5
−6	5	−9	5	−1	0
−7	0	−1	6	−14	14
−8	14	−28	20	−7	1

(Для семи лент таблица была бы аналогичной, однако строки с нечетными n были бы сдвинуты вправо на один столбец.) Тайна последовательности $a_0, a_{-2}, a_{-4}, \ldots = 1, 1, 2, 5, 14, \ldots$ мгновенно раскрывается специалистом по информатике, так как эта последовательность встречается в связи с очень большим числом рекурсивных алгоритмов (см., например, упр. 2.2.1–4 и формулу 2.3.4.4–(14)). Итак, мы предполагаем, что в случае T-лент

$$
\begin{aligned}
a_{-2n} &= \binom{2n}{n}\frac{1}{n+1} && \text{при } 0 \le n \le T-2;\\
a_{-2n-1} &= 0 && \text{при } 0 \le n \le T-3.
\end{aligned}
\tag{6}
$$

Чтобы проверить правильность этого предположения, достаточно показать, что (6) и (4) приводят к верным результатам для уровней 0 и 1. Для уровня 1 это очевидно, а для уровня 0 необходимо проверить, что

$$\binom{m}{0}a_0 - \binom{m+1}{2}a_{-2} + \binom{m+2}{4}a_{-4} - \binom{m+3}{6}a_{-6} + \cdots$$
$$= \sum_{k\ge 0}\binom{m+k}{2k}\binom{2k}{k}\frac{(-1)^k}{k+1} = \delta_{m0} \quad (7)$$

для $0 \le m \le T-2$. К счастью, эту сумму можно вычислить стандартными методами (это фактически пример 2 из раздела 1.2.6).

Теперь можно вычислить коэффициенты при $B(z) - q_1(z)A(z)$ и т. д. Рассмотрим, например, коэффициент при z^{2m} в $D(z) - q_3(z)A(z)$. Он равен

$$\sum_{k\ge 0}\binom{3+m+k}{2m+2k}(-1)^{m+k}a_{-2k} = \sum_{k\ge 0}\binom{3+m+k}{2m+2k}\binom{2k}{k}\frac{(-1)^{m+k}}{k+1}$$
$$= (-1)^m\left(\binom{2+m}{2m-1} - \binom{3+m}{2m}\right)$$
$$= (-1)^{m+1}\binom{2+m}{2m},$$

как следует из результата примера 3, приведенного в разделе 1.2.6. Таким образом, получены формулы

$$\begin{aligned} A(z) &= q_0(z)A(z), \\ B(z) &= q_1(z)A(z) - q_0(z), \qquad & C(z) &= q_2(z)A(z) - q_1(z), \\ D(z) &= q_3(z)A(z) - q_2(z), \qquad & E(z) &= q_4(z)A(z) - q_3(z). \end{aligned} \quad (8)$$

Кроме того, имеем $e_{n+1} = a_n$; следовательно, $zA(z) = E(z)$ и

$$A(z) = q_3(z)/\big(q_4(z) - z\big). \quad (9)$$

Производящие функции были выражены при помощи q-многочленов, поэтому желательно лучше изучить q. В этом отношении полезно упр. 1.2.9–15, так как оно дает выражение в замкнутом виде:

$$q_m(z) = \frac{\big((\sqrt{4-z^2}+iz)/2\big)^{2m+1} + \big((\sqrt{4-z^2}-iz)/2\big)^{2m+1}}{\sqrt{4-z^2}}. \quad (10)$$

Все упрощается, если теперь положить $z = 2\sin\theta$:

$$q_m(2\sin\theta) = \frac{(\cos\theta + i\sin\theta)^{2m+1} + (\cos\theta - i\sin\theta)^{2m+1}}{2\cos\theta} = \frac{\cos(2m+1)\theta}{\cos\theta}. \quad (11)$$

(Такое совпадение приводит к мысли, что многочлены $q_m(z)$ хорошо изучены в математике. Действительно, заглянув в соответствующие таблицы, мы видим, что $q_m(z)$, по существу, является многочленом Чебышева второго рода, а именно — $(-1)^m U_{2m}(z/2)$ в обычных обозначениях.)

Теперь можно определить корни знаменателя в (9): уравнение $q_4(2\sin\theta) = 2\sin\theta$ сводится к

$$\cos 9\theta = 2\sin\theta\cos\theta = \sin 2\theta.$$

Решения этого соотношения получаем, если только $\pm 9\theta = 2\theta + (2n - \frac{1}{2})\pi$; все такие θ дают корни знаменателя в (9) при условии, что $\cos\theta \ne 0$. (Если $\cos\theta = 0$,

то $q_m(\pm 2) = (2m+1)$ никогда не равно ± 2.) Следовательно, получаем восемь различных корней: $q_4(z) - z = 0$ при

$$2\sin\tfrac{-5}{14}\pi,\ 2\sin\tfrac{-1}{14}\pi,\ 2\sin\tfrac{3}{14}\pi;\ 2\sin\tfrac{-7}{22}\pi,\ 2\sin\tfrac{-3}{22}\pi,\ 2\sin\tfrac{1}{22}\pi,\ 2\sin\tfrac{5}{22}\pi,\ 2\sin\tfrac{9}{22}\pi.$$

Так как $q_4(z)$ — многочлен степени 8, значит, учтены все корни. Первые три из этих значений дают $q_3(z) = 0$, так что $q_3(z)$ и $q_4(z) - z$ имеют в качестве общего делителя многочлен третьей степени. Остальные пять корней управляют асимптотическим поведением коэффициентов $A(z)$, если разложить (9) на элементарные дроби.

Переходя к рассмотрению общего случая для T лент, положим $\theta_k = (4k+1)\pi/$ $/(4T-2)$. Производящая функция $A(z)$ для T лент каскадных чисел принимает вид

$$\frac{4}{2T-1}\sum_{-T/2<k<\lfloor T/2\rfloor}\frac{\cos^2\theta_k}{1-z/(2\sin\theta_k)} \tag{12}$$

(см. упр. 8); следовательно,

$$a_n = \frac{4}{2T-1}\sum_{-T/2<k<\lfloor T/2\rfloor}\cos^2\theta_k\left(\frac{1}{2\sin\theta_k}\right)^n. \tag{13}$$

Соотношения (8) приводят теперь к аналогичным формулам:

$$\begin{aligned} b_n &= \frac{4}{2T-1}\sum_{-T/2<k<\lfloor T/2\rfloor}\cos\theta_k\cos 3\theta_k\left(\frac{1}{2\sin\theta_k}\right)^n,\\ c_n &= \frac{4}{2T-1}\sum_{-T/2<k<\lfloor T/2\rfloor}\cos\theta_k\cos 5\theta_k\left(\frac{1}{2\sin\theta_k}\right)^n,\\ d_n &= \frac{4}{2T-1}\sum_{-T/2<k<\lfloor T/2\rfloor}\cos\theta_k\cos 7\theta_k\left(\frac{1}{2\sin\theta_k}\right)^n \end{aligned} \tag{14}$$

и т. д. В упр. 9 показано, что эти уравнения справедливы для всех $n \geq 0$, а не только для больших n. В каждой сумме член с $k = 0$ значительно превосходит все остальные члены, особенно если n достаточно велико. Следовательно, "отношение роста" есть

$$\frac{1}{2\sin\theta_0} = \frac{2}{\pi}T - \frac{1}{\pi} + \frac{\pi}{48T} + O(T^{-2}). \tag{15}$$

Каскадная сортировка впервые была исследована У. К. Картером (см. W. C. Carter, *Proc. IFIP Congress* (1962), 62–66), который получил численные результаты для небольших значений T, и Дэвидом Э. Фергюсоном [см. E. Ferguson, *CACM* **7** (1964), 297], который открыл первые два члена в асимптотическом поведении (15) отношения роста. Летом 1964 года Р. У. Флойд (R. W. Floyd) получил явный вид $1/(2\sin\theta_0)$ для отношения роста, так что точные формулы могли быть использованы для всех T. Глубокий анализ каскадных чисел был независимо выполнен Дж. Н. Рэйни (см. G. N. Raney, *Canadian J. Math.* **18** (1966), 332–349), который столкнулся с ними совершенно другим путем, не имея дела с сортировкой. Рэйни заметил принцип "отношения диагоналей", представленный на рис. 73, и вывел много других интересных свойств этих чисел. Флойд и Рэйни в своих доказательствах оперировали матрицами (упр. 6).

Модификация каскадной сортировки. Если добавить еще одну ленту, то почти все операции перемотки в процессе каскадной сортировки можно совместить. Например, можно выполнить слияние T1–T5 на T7, T1–T4 на T6, T1–T3 на T5 (которая к этому моменту уже перемотана), T1–T2 на T4 и начать следующий проход, когда на T4 будет перемотано сравнительно немного данных. Эффективность этого процесса можно предсказать на основании изложенного выше анализа каскадного метода (подробности приводятся в разделе 5.4.6).

Схема "компромиссного слияния", которая включает многофазную и каскадную схемы как частные случаи, была предложена Д. Э. Кнутом в *CACM* **6** (1963), 585–587. Каждая фаза состоит из $(T-1)$-, $(T-2)$-, ..., P-путевого слияний, где P — любое фиксированное число между 1 и $T-1$. Если $P = T-1$, значит, это многофазный метод; если $P = 1$, это чистый каскадный метод; если $P = 2$, это каскадный метод без фаз копирования. Анализ такой схемы был проделан в работах C. E. Radke, *IBM Systems J.* **5** (1966), 226–247, и W. H. Burge, *Proc. IFIP Congress* (1971), **1**, 454–459. Бурж нашел производящую функцию $\sum T_n(x)z^n$ для каждого (P,T)-компромиссного слияния, обобщающую соотношение 5.4.2–(16). Он показал, что наилучшее значение P (с точки зрения наименьшего числа обрабатываемых начальных серий) как функции от S при $S \to \infty$ (если непосредственно использовать схему распределения и пренебречь временем перемотки) есть соответственно $(2,3,3,4,4,4,3,3,4)$ при $T = (3,4,5,6,7,8,9,10,11)$. Эти значения P с ростом T сильнее отклоняются в сторону каскадного, а не многофазного метода, и оказывается, что компромиссное слияние никогда не станет существенно лучше каскадного. С другой стороны, при оптимальном выборе уровней и распределении фиктивных серий, как описано в разделе 5.4.2, чистый многофазный метод кажется наилучшим среди всех компромиссных методов слияния. К сожалению, оптимальное распределение сравнительно трудно реализовать.

В работе Th. L. Johnsen, *BIT* **6** (1966), 129–143, исследовано сочетание сбалансированного и многофазного слияний; модификация сбалансированного слияния с совмещением перемоток предложена в работе M. A. Гоетца (M. A. Goetz), *Digital Computer User's Handbook*, M. Klerer, G. A. Korn (New York: McGraw-Hill, 1967), 1.311–1.312. Можно представить себе и многие другие гибридные схемы.

УПРАЖНЕНИЯ

1. [*10*] Используя табл. 1, сравните каскадное слияние с описанной в разделе 5.4.2 версией многофазного слияния с "расщеплением лент". Какой метод лучше? (Временем перемотки пренебречь.)

▶ **2.** [*22*] Сравните метод каскадной сортировки с тремя лентами, использующий алгоритм C, и метод многофазной сортировки с тремя лентами, использующий алгоритм 5.4.2D. Какие сходства и различия вы заметите?

3. [*23*] Составьте таблицу, показывающую, что происходит при сортировке на шести лентах ста начальных серий при помощи алгоритма C.

4. [*M20*] (Дж. Н. Рэйни (G. N. Raney).) "Каскадное распределение n-го уровня" — мультимножество, определенное следующим образом (для шести лент): $\{1,0,0,0,0\}$ есть каскадное распределение 0-го уровня; если на последующих уровнях $\{a,b,c,d,e\}$ — каскадное распределение n-го уровня, то $\{a+b+c+d+e,\ a+b+c+d,\ a+b+c,\ a+b,\ a\}$ будет каскадным распределением $(n+1)$-го уровня. (Так как мультимножество не упорядочено,

из единственного распределения n-го уровня можно образовать до 5! различных распределений $(n+1)$-го уровня.)

a) Докажите, что *любое* мультимножество $\{a, b, c, d, e\}$ из взаимно простых чисел является каскадным распределением n-го уровня при некотором n.

b) Докажите, что распределение, используемое в каскадной сортировке, *оптимально* в том смысле, что если $\{a, b, c, d, e\}$ — любое распределение n-го уровня, причем $a \geq b \geq c \geq d \geq e$, то будем иметь $a \leq a_n$, $b \leq b_n$, $c \leq c_n$, $d \leq d_n$, $e \leq e_n$, где $(a_n, b_n, c_n, d_n, e_n)$ — распределение, определенное в (1).

▶ **5.** [*20*] Докажите, что каскадные числа, определенные в (1), удовлетворяют закону

$$a_k a_{n-k} + b_k b_{n-k} + c_k c_{n-k} + d_k d_{n-k} + e_k e_{n-k} = a_n \qquad \text{при } 0 \leq k \leq n.$$

[*Указание.* Для лучшего понимания этого соотношения рассмотрите, сколько серий различной длины выводится в течение k-го прохода полной каскадной сортировки.]

6. [*M20*] Найдите матрицу Q размера 5×5, такую, что первая строка Q^n содержит каскадные числа для шести лент $a_n\, b_n\, c_n\, d_n\, e_n$ при всех $n \geq 0$.

7. [*M20*] При условии, что каскадное слияние применяется к точному распределению a_n начальных серий, найдите формулу для величины сэкономленной работы, когда исключается однопутевое слияние.

8. [*HM23*] Выведите формулу (12).

9. [*HM26*] Выведите формулы (14).

▶ **10.** [*M28*] Вместо системы (4) для изучения каскадных чисел воспользуйтесь в качестве исходных тождествами

$$\begin{aligned} e_n &= a_{n-1} &&= lft\tbinom{1}{1}a_{n-1}, \\ d_n &= 2a_{n-1} - e_{n-2} &&= lft\tbinom{2}{1}a_{n-1} - \tbinom{3}{3}a_{n-3}, \\ c_n &= 3a_{n-1} - d_{n-2} - 2e_{n-2} &&= lft\tbinom{3}{1}a_{n-1} - \tbinom{4}{3}a_{n-3} - \tbinom{5}{5}a_{n-5} \end{aligned}$$

и т. д. Полагая

$$r_m(z) = \binom{m}{1}z - \binom{m+1}{3}z^3 + \binom{m+2}{5}z^5 - \cdots,$$

выразите через эти r-многочлены $A(z)$, $B(z)$ и т. д.

11. [*M38*] Пусть

$$f_m(z) = \sum_{k=0}^{m} \binom{\lfloor (m+k)/2 \rfloor}{k} (-1)^{\lceil k/2 \rceil} z^k.$$

Докажите, что производящая функция $A(z)$ для T-ленточных каскадных чисел есть

$$f_{T-3}(z)/f_{T-1}(z),$$

причем числитель и знаменатель этого выражения не имеют общих делителей.

12. [*M40*] Докажите, что схема распределения Фергюсона оптимальна в том смысле, что при любом другом методе размещения фиктивных серий, удовлетворяющем (2), во время первого прохода будет обрабатываться не меньше начальных серий *при условии*, что во время этого прохода используется стратегия шагов C7–C9.

13. [*40*] В тексте раздела предполагается бóльшую часть времени перемотки совмещать путем добавления дополнительной ленты. Попытайтесь развить эту идею. (Например, схема, приведенная в тексте, включает ожидание перемотки ленты T4. Не будет ли лучше исключить T4 из первой фазы слияния следующего прохода?)

*5.4.4. Чтение ленты в обратном направлении

Многие накопители на магнитных лентах позволяют читать ленту в направлении, противоположном тому, в котором осуществлялась запись. Рассмотренные ранее схемы слияния предполагают запись информации на ленту в прямом направлении, перемотку ленты, ее чтение в прямом направлении и еще одну перемотку. (Файлы на ленте, следовательно, ведут себя, как очереди "первым включается — первым исключается".) Чтение в обратном направлении (в дальнейшем будем для краткости называть эту операцию обратным чтением) позволяет обойтись без перемотки: запись на ленту выполняется в прямом направлении и считывается в обратном. (В этом случае файлы ведут себя, как стеки, поскольку здесь действует правило "последним включается — первым исключается".) Схемы сбалансированного, многофазного и каскадного слияний можно приспособить для обратного чтения. Основное отличие состоит в том, что *слияние изменяет порядок серий*, если считывание происходит в прямом направлении, а запись — в обратном. Если две серии находятся на ленте в порядке возрастания, то их можно слить, выполняя считывание в обратном направлении, однако при этом серии расположатся в порядке убывания. Полученные таким образом нисходящие серии станут восходящими на следующем проходе; следовательно, алгоритм слияния должен уметь работать с сериями обоих направлений. Программисту, впервые столкнувшемуся с обратным чтением, может показаться, что он стоит на голове!

В качестве примера обратного чтения рассмотрим процесс слияния восьми начальных серий с использованием *сбалансированного* слияния на четырех лентах. Записать наши действия можно следующим образом.

	T1	T2	T3	T4	
Проход 1	$A_1A_1A_1A_1$	$A_1A_1A_1A_1$	—	—	Начальное распределение
Проход 2	—	—	D_2D_2	D_2D_2	Слияние на T3 и T4
Проход 3	A_4	A_4	—	—	Слияние на T1 и T2
Проход 4	—	—	D_8	—	Окончательное слияние на T3

Здесь A_r обозначает серию, имеющую относительную длину r и расположенную в порядке возрастания, если лента читается в прямом направлении, как в предыдущих примерах; D_r — аналогичное обозначение для нисходящей серии длиной r. Во время второго прохода восходящие серии становятся нисходящими. Они оказываются нисходящими при вводе, так как мы читаем ленты T1 и T2 в обратном направлении. Серии вновь изменяют ориентацию на третьем проходе.

Заметим, что описанный процесс завершается формированием на ленте T3 результата в порядке *убывания*. Если это плохо (что зависит от того, должен ли результат читаться в обратном направлении или же лента, содержащая его, должна быть снята и отложена для использования в дальнейшем), можно скопировать его на другую ленту, изменив направление. Более быстрым способом была бы перемотка T1 и T2 после третьего прохода; при этом во время четвертого прохода получается A_8. Еще лучше было бы начать с восьми *нисходящих* серий на первом проходе, так как при этом поменялись бы местами все A и D. Однако для сбалансированного слияния 16 начальных серий потребовалось бы, чтобы начальные серии были восходящими, но поскольку обычно заранее неизвестно, сколько начальных серий будет

образовано, необходимо выбрать одно постоянное направление. Следовательно, идея перемотки после третьего прохода является наилучшей.

Каскадное слияние преобразуется таким же способом. Рассмотрим, например, сортировку 14 начальных серий на четырех лентах.

	T1	T2	T3	T4
Проход 1	$A_1A_1A_1A_1A_1A_1$	$A_1A_1A_1A_1A_1$	$A_1A_1A_1$	—
Проход 2	—	D_1	D_2D_2	$D_3D_3D_3$
Проход 3	A_6	A_5	A_3	—
Проход 4	—	—	—	D_{14}

Как и ранее, можно получить A_{14} вместо D_{14}, если перемотать T1, T2, T3 непосредственно перед последним проходом. Заметим, что это "чистое" каскадное слияние в том смысле, что все однопутевые слияния выполнены явным образом. Если бы мы запретили операции копирования, как в алгоритме 5.4.3C, то после второго прохода столкнулись бы с ситуацией

$$A_1 \qquad — \qquad D_2D_2 \qquad D_3D_3D_3$$

В этом случае невозможно продолжать работу, используя трехпутевое слияние, так как нельзя сливать серии противоположных направлений! Можно было бы избежать копирования T1 на T2, если перемотать T1 и начать ее считывание в прямом направлении на следующей фазе слияния (в то время как T3 и T4 читаются в обратном направлении). Но тогда пришлось бы вновь перемотать T1 после слияния, так что данный прием заменяет одно копирование двумя перемотками.

Таким образом, метод распределения алгоритма 5.4.3C при обратном чтении не столь эффективен, как при прямом чтении; временные затраты резко возрастают каждый раз, когда число начальных серий проходит через "точное" каскадное распределение. Чтобы получить более "гладкий" переход между точными каскадными распределениями, можно использовать иной метод распределения (см. упр. 17).

Обратное чтение в многофазном слиянии. На первый взгляд (и даже на второй и третий), схема многофазного слияния кажется совершенно неподходящей для обратного чтения. Предположим, например, что имеется 13 начальных серий и три ленты.

	T1	T2	T3
Фаза 1	$A_1A_1A_1A_1A_1$	$A_1A_1A_1A_1A_1A_1A_1A_1$	—
Фаза 2	—	$A_1A_1A_1$	$D_2D_2D_2D_2D_2$

Здесь мы оказываемся в тупике; можно было бы перемотать T2 или T3 и затем читать их в прямом направлении, в то время как остальные ленты — в обратном. Но это значительно запутало бы дело и преимущества от обратного чтения были бы невелики.

Остроумный выход из сложившейся ситуации состоит в том, чтобы *чередовать направления серий на каждой ленте*. Тогда слияние может происходить вполне согласовано.

	T1	T2	T3
Фаза 1	$A_1D_1A_1D_1A_1$	$D_1A_1D_1A_1D_1A_1D_1A_1$	—
Фаза 2	—	$D_1A_1D_1$	$D_2A_2D_2A_2D_2$
Фаза 3	$A_3D_3A_3$	—	D_2A_2
Фаза 4	A_3	D_5A_5	—
Фаза 5	—	D_5	D_8
Фаза 6	A_{13}	—	—

Этот принцип был кратко упомянут Р. Л. Гилстадом (R. L. Gilstad) в его ранней статье о многофазном слиянии. Более полное описание приводится в *CACM* **6** (1963), 220–223.

Рассматриваемый $ADA\ldots$-метод прекрасно подходит для многофазного слияния с *любым* числом лент; можно показать, что A и D согласуются соответствующим образом на каждой фазе, но только при условии, что на начальном проходе чередующиеся серии A и D должны быть сформированы на каждой ленте и что каждая лента заканчивается серией A (или каждая лента заканчивается серией D). Так как последняя серия, записываемая в файл вывода во время одной фазы, имеет направление, противоположное направлению последней использованной серии из файла ввода, то и на следующей фазе серии всегда будут иметь надлежащую ориентацию. Далее, как в упр. 5.4.2–13, большинство точных фибоначчиевых распределений требует *нечетного* числа серий на одной ленте (окончательной выводной ленте) и *четного* числа серий на всех остальных лентах. Если T1 предназначена для вывода конечного результата, значит, можно гарантировать, что все ленты будут заканчиваться серией A, если ленту T1 начнем с A, а все остальные ленты — с D. Можно использовать метод распределения, аналогичный алгоритму 5.4.2D, изменив его таким образом, чтобы распределение на каждом уровне имело в качестве выводной ленту T1. (Мы пропускаем уровни $1, T+1, 2T+1, \ldots$, так как на них окончательной выводной лентой является первоначально пустая лента.) Например, в случае для шести лент вместо 5.4.2–(1) можно использовать следующее распределение серий.

Уровень	T1	T2	T3	T4	T5	Сумма	Окончательная выводная лента	
0	1	0	0	0	0	1	T1	
2	1	2	2	2	2	9	T1	
3	3	4	4	4	2	17	T1	
4	7	8	8	6	4	33	T1	(1)
5	15	16	14	12	8	65	T1	
6	31	30	28	24	16	129	T1	
8	61	120	116	108	92	497	T1	

Таким образом, на T1 всегда помещается нечетное число серий, тогда как на ленты с T2 по T5 — четные числа (в порядке убывания, чтобы упростить присвоение фиктивных серий). Такое распределение имеет то преимущество, что окончательная выводная лента известна заранее независимо от числа начальных серий, которые придется обрабатывать. Оказывается (см. упр. 3), что если используется эта схема, то результат всегда будет находиться на T1 в порядке *возрастания*.

Другой способ распределения для многофазной схемы с обратным чтением был предложен в статье D. T. Goodwin, J. L. Venn, *CACM* **7** (1964), 315. Можно распределять серии, почти как в алгоритме 5.4.2D, начиная с D-серии на каждой ленте. Когда ввод исчерпан, мы представляем себе фиктивную A-серию расположенной в начале единственной "нечетной" ленты, если только не достигнуто распределение со всеми нечетными числами. Остальные фиктивные серии мы представляем себе расположенными в конце лент или сгруппированными в пары в середине. Вопрос об оптимальном размещении фиктивных серий анализируется ниже, в упр. 5.

Оптимальные схемы слияния. До сих пор мы обсуждали различные схемы слияния с лентами, не пытаясь найти метод, "наилучший из возможных". Определение оптимальной схемы кажется особенно сложным в случае прямого чтения, при котором влияние времени перемотки и времени слияния с трудом поддается анализу. С другой стороны, если слияние осуществляется посредством обратного чтения и прямой записи, то, по существу, все перемотки устраняются и появляется возможность довольно хорошо формализовать оптимизацию способов слияния. Ричард М. Карп (Richard M. Karp) предложил несколько интересных подходов к решению этой задачи, и мы завершаем настоящий раздел обсуждением развитой им теории.

Для начала необходим более удобный способ описания схем слияния вместо довольно таинственных таблиц "содержимого лент", которые использовались выше. Карп предложил два таких способа — *векторное представление схемы слияния* и *представление в виде дерева*. Оба представления можно с успехом использовать, и мы опишем их по очереди.

Векторное представление схемы слияния состоит из последовательности "векторов слияния" $y^{(m)} \ldots y^{(1)}\, y^{(0)}$, каждый из которых имеет T компонент; $y^{(i)}$ изображает i-й с конца шаг слияния следующим образом:

$$y_j^{(i)} = \begin{cases} +1, & \text{если лента с номером } j \text{ является вводной для данного слияния;} \\ 0, & \text{если лента с номером } j \text{ не используется в данном слиянии;} \\ -1, & \text{если на ленту с номером } j \text{ выводится результат данного слияния.} \end{cases} \tag{2}$$

Таким образом, ровно одна компонента $y^{(i)}$ равна -1, остальные компоненты равны 0 и 1. Итоговый вектор $y^{(0)}$ — особый: это единичный вектор, имеющий 1 в позиции j (если окончательный рассортированный результат оказывается на устройстве j) и 0 в остальных позициях. Из данного определения следует, что векторная сумма

$$v^{(i)} = y^{(i)} + y^{(i-1)} + \cdots + y^{(0)} \tag{3}$$

представляет собой распределение серий на лентах непосредственно перед i-м с конца шагом слияния; причем на ленте j находится $v_j^{(i)}$ серий. В частности, по $v^{(m)}$ можно судить, сколько серий помещается на каждую ленту на начальном проходе распределения.

Три схемы слияния, описанные ранее в этом разделе в табличной форме, имеют следующие векторные представления.

Сбалансированная (T=4, S=8)	Каскадная (T=4, S=14)	Многофазная (T=3, S=13)
$v^{(7)} = (\ 4,\ 4,\ 0,\ 0)$	$v^{(10)} = (\ 6,\ 5,\ 3,\ 0)$	$v^{(12)} = (\ 5,\ 8,\ 0)$
$y^{(7)} = (+1, +1, -1,\ 0)$	$y^{(10)} = (+1, +1, +1, -1)$	$y^{(12)} = (+1, +1, -1)$
$y^{(6)} = (+1, +1,\ 0, -1)$	$y^{(9)} = (+1, +1, +1, -1)$	$y^{(11)} = (+1, +1, -1)$
$y^{(5)} = (+1, +1, -1,\ 0)$	$y^{(8)} = (+1, +1, +1, -1)$	$y^{(10)} = (+1, +1, -1)$
$y^{(4)} = (+1, +1,\ 0, -1)$	$y^{(7)} = (+1, +1, -1,\ 0)$	$y^{(9)} = (+1, +1, -1)$
$y^{(3)} = (-1,\ 0, +1, +1)$	$y^{(6)} = (+1, +1, -1,\ 0)$	$y^{(8)} = (+1, +1, -1)$
$y^{(2)} = (\ 0, -1, +1, +1)$	$y^{(5)} = (+1, -1,\ 0,\ 0)$	$y^{(7)} = (-1, +1, +1)$
$y^{(1)} = (+1, +1, -1,\ 0)$	$y^{(4)} = (-1, +1, +1, +1)$	$y^{(6)} = (-1, +1, +1)$
$y^{(0)} = (\ 0,\ 0,\ 1,\ 0)$	$y^{(3)} = (\ 0, -1, +1, +1)$	$y^{(5)} = (-1, +1, +1)$
	$y^{(2)} = (\ 0,\ 0, -1, +1)$	$y^{(4)} = (+1, -1, +1)$
	$y^{(1)} = (+1, +1, +1, -1)$	$y^{(3)} = (+1, -1, +1)$
	$y^{(0)} = (\ 0,\ 0,\ 0,\ 1)$	$y^{(2)} = (+1, +1, -1)$
		$y^{(1)} = (-1, +1, +1)$
		$y^{(0)} = (\ 1,\ 0,\ 0)$

Может показаться неудобным нумеровать данные векторы с конца так, чтобы $y^{(m)}$ появлялся первым, а $y^{(0)}$ — последним, но этот нестандартный способ нумерации более удобен при разработке теории. Для поиска оптимального метода неплохо сначала вывести рассортированный результат и представить себе, как его можно распределить на различные ленты (т. е. выполнить операцию, обратную слиянию). Затем следует распределить то, что получилось, и т. д., рассматривая последовательные распределения $v^{(0)}$, $v^{(1)}$, $v^{(2)}, \ldots$ в порядке, обратном тому, в котором они в действительности появляются в результате слияний в ходе сортировки. Фактически именно этот подход использовался нами при анализе многофазного и каскадного слияний.

Каждая схема слияния, очевидно, имеет векторное представление. И обратно, как легко видеть, последовательность векторов $y^{(m)} \ldots y^{(1)}\, y^{(0)}$ соответствует реальной схеме слияния тогда и только тогда, когда выполняются следующие три условия:

i) вектор $y^{(0)}$ является единичным;

ii) ровно одна компонента вектора $y^{(i)}$ равна -1; все остальные компоненты равны 0 или $+1$, $m \geq i \geq 1$;

iii) все компоненты вектора $y^{(i)} + \cdots + y^{(1)} + y^{(0)}$ неотрицательны, $m \geq i \geq 1$.

Информацию о схеме слияния можно представить в виде дерева. Мы строим дерево с одним внешним "листовым" узлом для каждой начальной серии и с одним внутренним узлом для каждой серии, полученной в результате слияния. Потомками любого внутреннего узла являются образующие его серии. Каждый внутренний узел помечается номером шага, на котором была образована соответствующая серия, при этом шаги нумеруются в обратном порядке, как в векторном представлении. Кроме того, ребра непосредственно над каждым узлом помечаются именем ленты, на которой находится данная серия. Например, три приведенные выше схемы имеют представления в виде дерева, изображенные на рис. 76, если мы назовем ленты A, B, C, D, а не T1, T2, T3, T4.

Это удобное и наглядное представление многих существенных свойств схемы слияния. Например, если серия на уровне 0 дерева (корень) должна быть восходящей, то серии на уровне 1 должны быть нисходящими, серии на уровне 2 — восходящими и т. д. Некоторая начальная серия является восходящей тогда и только

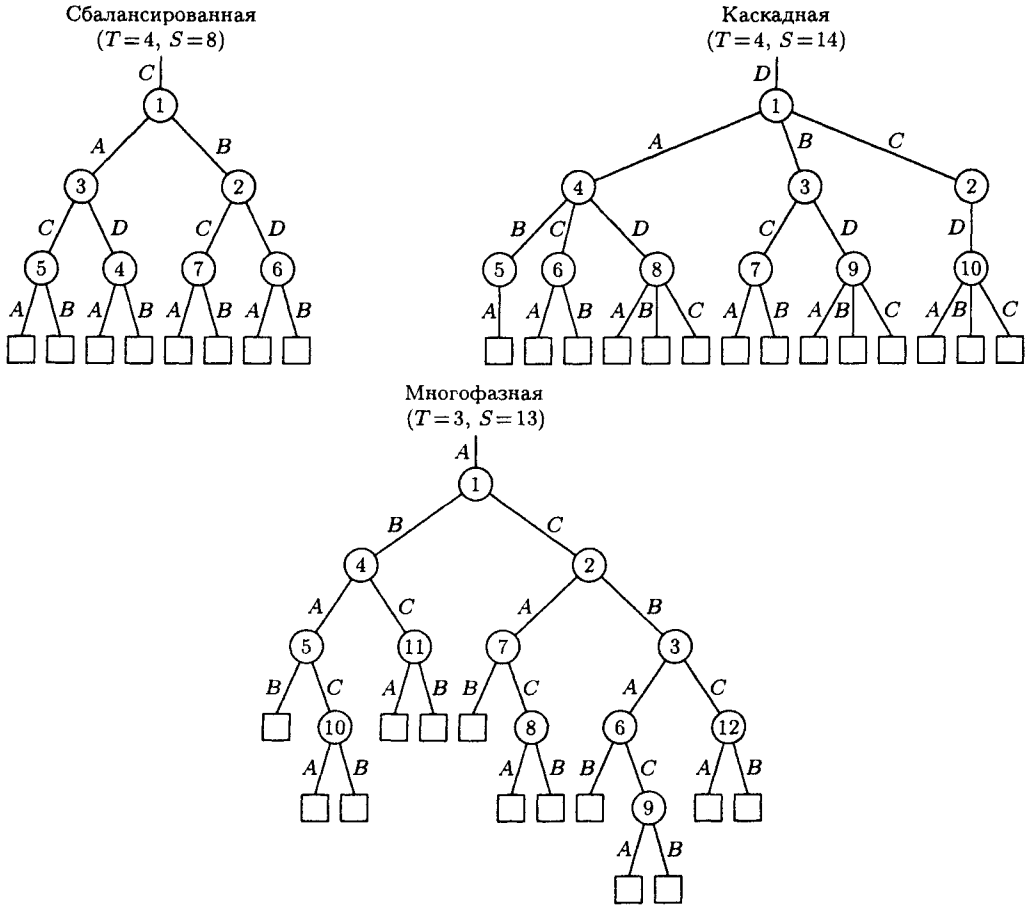


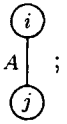
Рис. 76. Представления трех схем слияния в виде деревьев.

тогда, когда соответствующий внешний узел находится на уровне с четным номером. Далее, суммарное количество начальных серий, обрабатываемых при слиянии (не включая начальное распределение), в точности равно *длине внешнего пути* дерева, поскольку каждая начальная серия на уровне k обрабатывается ровно k раз.

Любая схема слияния имеет представление в виде дерева, но не каждое дерево определяет схему слияния. Дерево, внутренние узлы которого помечены числами от 1 до m и ребра которого помечены именами лент, изображает правильную схему слияния с обратным чтением тогда и только тогда, когда

a) никакие два ребра, смежные с одним внутренним узлом, не имеют одного и того же имени ленты;

b) если $i > j$ и если A есть имя ленты, то дерево не содержит конфигурации



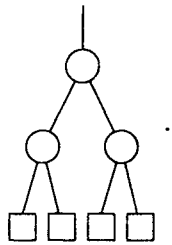
;

c) если $i < j < k < l$ и если A — имя ленты, то дерево не содержит таких пар

(4)

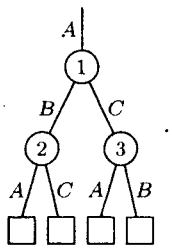
Условие (a) очевидно, так как вводные и выводные ленты слияния должны быть различны; условие (b) также очевидно. Условие "непересечения" (c) отражает характерное для операций обратного чтения ленты ограничение "последним включается — первым исключается". Серия, образованная на шаге k, должна быть удалена прежде серии, сформированной ранее на той же ленте; следовательно, конфигурации (4) невозможны. Нетрудно проверить, что любое помеченное дерево, удовлетворяющее условиям (a), (b) и (c), действительно соответствует некоторой схеме слияния с обратным чтением.

Если имеется T накопителей на магнитной ленте, то из условия (a) следует, что степень каждого внутреннего узла равна $T - 1$ или меньше. Присвоить подходящие метки всем таким деревьям не всегда возможно; например, если $T = 3$, то не существует схемы слияния с деревом вида



(5)

Такая форма дерева позволила бы найти оптимальную схему слияния, если бы можно было соответствующим образом присвоить номера шагов и номера лент, поскольку это единственное дерево с минимальной длиной внешнего пути среди деревьев, имеющих четыре внешних узла. Но в силу симметрии структуры этого дерева имеется, по существу, только один способ пометить его, соблюдая условия (a) и (b):



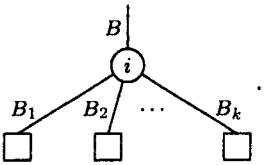
(6)

Однако при этом нарушается условие (c). Дерево, которое *может* быть помечено в соответствии с упомянутыми условиями с использованием T или меньше имен лент, называется T-*lifo*-деревом. Другой способ охарактеризовать все помеченные деревья, которые могут возникнуть из схемы слияния, состоит в рассмотрении метода "выращивания" всех подобных деревьев. Начнем с некоторого имени ленты, скажем

A, и с ростка дерева

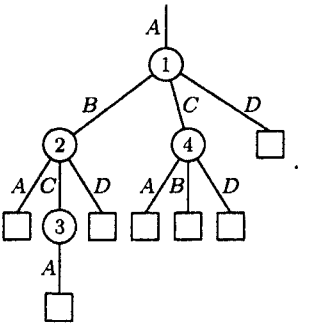
Шаг i роста дерева состоит в выборе различных имен лент $B, B_1, B_2, \ldots, B_k$ и *дальнейшей замене всего образованного* узла, соответствующего B,

от

до 

. (7)

Правило "последним образован — первым растет" в точности описывает способ построения представления в виде дерева непосредственно из векторного представления. Определение строго оптимальной T-ленточной схемы слияния, т. е. дерева, имеющего минимальную длину пути среди всех T-lifo-деревьев с данным числом внешних узлов, кажется весьма трудной задачей. Следующая неочевидная схема, например, представляет наилучший способ слияния семи начальных серий с помощью четырех лент и считывания в обратном направлении:



. (8)

По существу, для достижения оптимума необходимо однопутевое слияние! (См. упр. 8.) С другой стороны, не так уж трудно построить конструкции, *асимптотически* оптимальные для любого фиксированного T.

Пусть $K_T(n)$ — минимальная длина внешнего пути, достижимая в T-lifo-дереве с n внешними узлами. Используя теорию, развитую в разделе 2.3.4.5, несложно доказать, что

$$K_T(n) \geq nq - \lfloor ((T-1)^q - n)/(T-2) \rfloor, \qquad q = \lceil \log_{T-1} n \rceil, \tag{9}$$

так как это минимальная длина внешнего пути *любого* дерева с n внешними узлами и степенью любого узла $< T$. К настоящему моменту известны относительно немногие точные значения $K_T(n)$. Ниже приведены верхние оценки, которые, вероятно, точны.

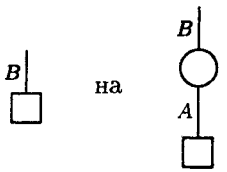
$n =$	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$K_3(n) \leq$	0	2	5	9	12	16	21	25	30	34	39	45	50	56	61
$K_4(n) \leq$	0	2	3	6	8	11	14	17	20	24	27	31	33	37	40

(10)

Карп обнаружил, что *любое* дерево с внутренними узлами степени $< T$ является *почти* T-lifo-деревом в том смысле, что оно может быть превращено в T-lifo путем замены некоторых внешних узлов однопутевыми слияниями. Фактически создать подходящую расстановку меток довольно просто. Пусть A — конкретное имя ленты. Необходимо выполнить следующие действия.

Шаг 1. Присвоить имена лент ребрам схемы дерева любым способом, совместимым с условием (a), которое приведено выше, однако так, чтобы специальное имя A использовалось только в крайнем слева ребре ветви.

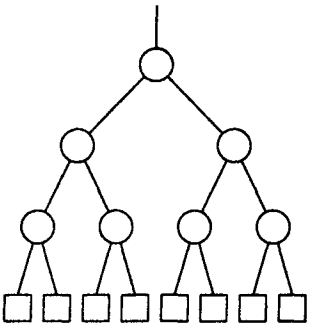
Шаг 2. Заменить каждый внешний узел вида



для любого $B \neq A$.

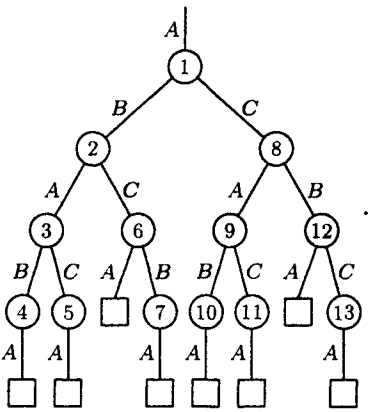
Шаг 3. Пронумеровать внутренние узлы в *прямом порядке.* Результатом будет расстановка меток, удовлетворяющая условиям (a), (b) и (c).

Например, если начать с дерева



(11)

и трех лент, то эта процедура расставит метки следующим образом:



(12)

Нетрудно проверить, что конструкция Карпа удовлетворяет дисциплине "последним образован — первым растет" в силу свойств прямого порядка (см. упр. 12).

Заметим, что результатом такого построения является схема слияния, в которой все начальные серии появляются на ленте A. Это предполагает следующую схему распределения и сортировки, которую можно назвать *слиянием в прямом порядке.*

P1. Распределить начальные серии на ленту A, пока ввод не будет исчерпан. Пусть S — число всех начальных серий.

P2. Выполнить описанное выше построение, используя $(T-1)$-арное дерево с S внешними узлами и минимальной длиной пути, и получить T-lifo-дерево, длина внешнего пути которого превышает нижнюю границу (9) не более чем на S.

P3. Слить серии в соответствии с этой схемой. ▮

Результат в указанной схеме получается на какой угодно ленте. Но *эта схема имеет один серьезный изъян.* (Видит ли читатель, что именно здесь будет неправильно работать?) Дело в том, что схема требует, чтобы первоначально одни серии на ленте A были восходящими, а другие — нисходящими в зависимости от того, где появляется соответствующий внешний узел: на нечетном или четном уровне. Не зная S заранее, эту проблему можно разрешить путем копирования серий, которые должны быть нисходящими, на вспомогательную ленту (или ленты) непосредственно перед тем, как они потребуются. Тогда суммарное количество операций, измеряемое в длинах начальных серий, окажется равным

$$S\log_{T-1} S + O(S). \tag{13}$$

Таким образом, слияние в прямом порядке определенно лучше многофазного или каскадного при $S \to \infty$. В действительности оно асимптотически *оптимально*, так как (9) показывает, что $S\log_{T-1} S + O(S)$ — это наименьшая оценка времени, на которую мы вообще можем надеяться при работе с T лентами. С другой стороны, для сравнительно небольших значений S, обычно встречающихся на практике, слияние в прямом порядке весьма неэффективно; многофазный или каскадный метод проще и быстрее, если S относительно мало. Возможно, удастся изобрести простую схему распределения и слияния, которая сравнима с многофазной и каскадной при небольших S и асимптотически оптимальна при больших S.

Ниже, в упражнениях части 2, демонстрируется, как Карп аналогичным образом поставил вопрос для слияния с *прямым чтением.* Теория оказывается в этом случае значительно более сложной, хотя были получены некоторые весьма интересные результаты.

УПРАЖНЕНИЯ (часть 1)

1. [*17*] При слиянии с прямым чтением часто удобно отмечать конец каждой серии на ленте путем добавления искусственной "концевой" записи с ключом $+\infty$. Как следует видоизменить этот метод при обратном чтении?

2. [*20*] Будут ли столбцы таблицы, аналогичной табл. 1, всегда неубывающими или возникают ситуации, когда приходится "вычитать" серии с некоторой ленты при переходе от одного уровня к другому?

▶ **3.** [*20*] Докажите, что при использовании метода многофазного слияния, описанного для распределения (1), после завершения сортировки на ленте T1 всегда оказывается и серия A, если первоначально на T1 было $ADA\ldots$, а на лентах T2–T5 было $DAD\ldots$.

4. [*M22*] Как вы оцениваете идею выполнения многофазного слияния с обратным чтением после распределения всех серий в порядке *возрастания*, если предположить, что все позиции D первоначально заполнены фиктивными сериями?

▶ **5.** [*23*] Какие формулы для цепочек слияния чисел вместо (8)–(11) из раздела 5.4.2 будут справедливы для многофазного слияния с обратным чтением? Оцените число слияний для распределения пятого уровня на шести лентах, нарисовав диаграмму, аналогичную показанной на рис. 71, (a).

6. [*07*] Каково векторное представление схемы слияния, представлением которой в виде дерева является (8)?

7. [*16*] Нарисуйте представление в виде дерева схемы слияния с обратным чтением, определенной такой последовательностью векторов:

$$
\begin{array}{lll}
v^{(33)} = (\ 20,\ \ 9,\ \ 5) & y^{(22)} = (+1,-1,+1) & y^{(10)} = (+1,+1,-1) \\
y^{(33)} = (+1,-1,+1) & y^{(21)} = (-1,+1,+1) & y^{(9)} \ = (+1,-1,+1) \\
y^{(32)} = (+1,+1,-1) & y^{(20)} = (+1,+1,-1) & y^{(8)} \ = (+1,+1,-1) \\
y^{(31)} = (+1,+1,-1) & y^{(19)} = (-1,+1,+1) & y^{(7)} \ = (+1,+1,-1) \\
y^{(30)} = (+1,+1,-1) & y^{(18)} = (+1,+1,-1) & y^{(6)} \ = (+1,+1,-1) \\
y^{(29)} = (+1,-1,+1) & y^{(17)} = (+1,+1,-1) & y^{(5)} \ = (-1,+1,+1) \\
y^{(28)} = (-1,+1,+1) & y^{(16)} = (+1,+1,-1) & y^{(4)} \ = (+1,-1,+1) \\
y^{(27)} = (+1,-1,+1) & y^{(15)} = (+1,+1,-1) & y^{(3)} \ = (-1,+1,+1) \\
y^{(26)} = (+1,-1,+1) & y^{(14)} = (+1,-1,+1) & y^{(2)} \ = (+1,-1,+1) \\
y^{(25)} = (+1,+1,-1) & y^{(13)} = (+1,-1,+1) & y^{(1)} \ = (-1,+1,+1) \\
y^{(24)} = (+1,-1,+1) & y^{(12)} = (-1,+1,+1) & y^{(0)} \ = (\ \ 1,\ \ 0,\ \ 0) \\
y^{(23)} = (+1,-1,+1) & y^{(11)} = (+1,+1,-1) &
\end{array}
$$

8. [*23*] Докажите, что (8) — оптимальный способ слияния с обратным чтением при $S = 7$ и $T = 4$ и что все методы, избегающие однопутевого слияния, хуже.

9. [*M22*] Докажите справедливость нижней оценки в (9).

10. [*41*] При помощи компьютера составьте таблицу точных значений $K_T(n)$.

▶ **11.** [*20*] Верно ли утверждение, что для любой схемы слияния с обратным чтением, не использующей ничего, кроме $(T-1)$-путевого слияния, необходимо, чтобы серии на каждой ленте чередовались: $ADAD\ldots$ (т. е. такая схема не будет работать, если две соседние серии окажутся одинаково упорядоченными)?

12. [*22*] Докажите, что конструкция с прямым порядком Карпа всегда порождает помеченное дерево, удовлетворяющее условиям (a), (b) и (c).

13. [*16*] Улучшите процедуру (12), сделайте ее более эффективной, удалив как можно больше однопутевых слияний, однако так, чтобы прямой порядок все еще приводил к правильной расстановке меток у внутренних узлов.

14. [*40*] Придумайте алгоритм, который выполняет слияние в прямом порядке без явного построения дерева на шагах P2 и P3, используя только $O(\log S)$ слов памяти для управления слиянием.

15. [*M39*] Конструкция Карпа с прямым порядком порождает деревья с однопутевым слиянием в некоторых терминальных узлах. Докажите, что если $T = 3$, то можно построить асимптотические оптимальные 3-lifo-деревья, в которых используется только двухпутевое слияние.

Другими словами, пусть $\hat{K}_T(n)$ будет минимальной длиной внешнего пути среди всех T-lifo-деревьев с n внешними узлами, таких, что каждый внутренний узел имеет степень $T-1$. Докажите, что $\hat{K}_3(n) = n \lg n + O(n)$.

16. [*M46*] (Сохраняются обозначения, принятые в упр. 15.) Верно ли, что $\hat{K}_T(n) = n \log_{T-1} n + O(n)$ для *всех* $T \geq 3$, если $n \equiv 1$ (по модулю $T-2$)?

▸ **17.** [*28*] (Ричард Д. Пратт (Richard D. Pratt).) Чтобы получить возрастающий порядок в каскадном слиянии с обратным чтением, можно потребовать *четного* числа проходов слияния. Таким образом, предполагается, что метод начального распределения несколько отличен от метода, приведенного в алгоритме 5.4.3C.

a) Измените 5.4.3–(1) так, чтобы были представлены только точные распределения, для которых требуется четное число проходов слияния.

b) Постройте схему начального распределения, осуществляющую интерполяцию между этими точными распределениями. [Иначе говоря, если число начальных серий находится между точными распределениями, то желательно слить некоторые (но не все) серии дважды, чтобы получить точное распределение.]

▸ **18.** [*M38*] Предположим, что имеется T ленточных устройств для некоторого $T \geq 3$ и что на T1 содержится N записей, в то время как остальные ленты пусты. Возможно ли обратить порядок записей на ленте T1 за число шагов, меньшее $\Omega(N \log N)$, *без* обратного чтения? (Эта операция является, конечно, тривиальной, если допускается обратное чтение.) В упр. 5.2.5–14 приводится класс таких алгоритмов, которые, однако, *тратят* порядка $N \log N$ шагов.

УПРАЖНЕНИЯ (часть 2)

В следующих упражнениях теория слияния на лентах с прямым чтением распространяется на случай, когда каждая лента действует, как очередь, а не как стек. Схему слияния можно представить последовательностью векторов $y^{(m)} \ldots y^{(1)} y^{(0)}$ точно так же, как в тексте раздела, но когда мы преобразуем векторное представление в представление в виде дерева, правило "последним образован — первым растет" заменяется правилом "*первым* образован — первым растет". Таким образом, недопустимая конфигурация (4) должна быть заменена конфигурацией

(4′)

Дерево, которое может быть помечено так, чтобы изображались слияния с прямым чтением на T лентах, называется T-fifo-деревом по аналогии с термином "T-lifo" в случае обратного чтения.

Если ленты можно прочесть в обратном направлении, значит, они образуют очень хорошие стеки. Но, к сожалению, они не могут образовать очень хорошие универсальные очереди. Если в произвольном порядке записывать и считывать по правилу "первым включается — первым исключается", то придется тратить много времени на переход от одной части ленты к другой. Хуже того, мы вскоре проскочим конец ленты! Здесь возникает та же проблема, что и в случае выхода очереди за границу памяти [см. соотношения 2.2.2–(4) и (5)], но решение в виде 2.2.2–(6) и (7) не применимо к лентам, так как их нельзя закольцевать. Поэтому дерево будем называть *сильным* T-*fifo*-деревом, если оно может быть помечено так, чтобы соответствующая схема слияния использовала ленты, подчиняясь особой дисциплине: "записать, перемотать, прочитать все, перемотать; записать, перемотать, прочитать все, перемотать и т. д.".

▶ **19.** [*22*] (Р. М. Карп.) Найдите бинарное дерево, которое не является 3-fifo.

▶ **20.** [*22*] Сформулируйте условие "сильного T-fifo-дерева" в терминах достаточно простого правила относительно недопустимых конфигураций меток лент, аналогичного (4′).

21. [*18*] Нарисуйте представление в виде дерева схемы слияния с прямым чтением, определенной посредством векторов в упр. 7. Можно ли считать его сильным 3-fifo-деревом?

22. [*28*] (Р. М. Карп.) Докажите, что представления в виде дерева многофазного и каскадного слияний с точным распределением полностью одинаковы как в случае обратного чтения, так и в случае прямого чтения, за исключением номеров, которыми помечены внутренние узлы. Найдите более широкий класс векторных представлений схем слияния, для которых верно сказанное.

23. [*24*] (Р. М. Карп.) Будем говорить, что серия $y^{(q)} \dots y^{(r)}$ схемы слияния является *стадией*, если никакая выводная лента не используется в дальнейшем как вводная, т. е. если не существует i, j, k, таких, что $q \geq i > k \geq r$ и $y_j^{(i)} = -1$, $y_j^{(k)} = +1$. Назначение этого упражнения — доказать, что *при каскадном слиянии минимизируется число стадий* среди всех схем слияния с тем же числом лент и начальных серий.

Удобно ввести некоторые обозначения. Будем писать $v \to w$, если v и w такие T-векторы, что существует схема слияния, которая на своей первой стадии переводит w в v (т. е. существует схема слияния $y^{(m)} \dots y^{(0)}$, такая, что $y^{(m)} \dots y^{(l+1)}$ является стадией, $w = y^{(m)} + \dots + y^{(0)}$ и $v = y^{(l)} + \dots + y^{(0)}$.) Будем писать $v \preceq w$, если v и w — T-векторы, такие, что сумма наибольших k элементов вектора $v \leq$ не превышает суммы наибольших k элементов вектора w при $1 \leq k \leq T$. Так, например, $(2,1,2,2,2,1) \preceq (1,2,3,0,3,1)$, поскольку $2 \leq 3$, $2+2 \leq 3+3$, $\dots$, $2+2+2+2+1+1 \leq 3+3+2+1+1+0$. Наконец, если $v = (v_1, \dots, v_T)$, то пусть $C(v) = (s_T, s_{T-2}, s_{T-3}, \dots, s_1, 0)$, где s_k есть сумма наибольших k элементов векторов v.

a) Докажите, что $v \to C(v)$.

b) Докажите, что $v \preceq w$ влечет $C(v) \preceq C(w)$.

c) Считая известным результат упр. 24, докажите, что при каскадном слиянии минимизируется число стадий.

24. [*M35*] Используя обозначения, принятые в упр. 23, докажите, что $v \to w$ влечет $w \preceq C(v)$.

25. [*M36*] (Р. М. Карп.) Будем говорить, что сегмент $y^{(q)} \dots y^{(r)}$ схемы слияния является *фазой*, если ни одна из лент не используется и для ввода, и для вывода, т. е. если не существует i, j, k, таких, что $q \geq i \geq r$, $q \geq k \geq r$, $y_j^{(i)} = +1$, и $y_j^{(k)} = -1$. Назначение этого упражнения — исследовать схему слияния, которая минимизирует число фаз. Будем писать $v \Rightarrow w$, если w может быть преобразовано в v за одну фазу (ср. с подобным обозначением, введенным в упр. 23), и пусть $D_k(v) = (s_k + t_{k+1}, s_k + t_{k+2}, \dots, s_k + t_T, 0, \dots, 0)$, где t_j обозначает j-й в порядке убывания элемент v и $s_k = t_1 + \dots + t_k$.

a) Докажите, что $v \Rightarrow D_k(v)$ при $1 \leq k < T$.

b) Докажите, что из $v \preceq w$ следует $D_k(v) \preceq D_k(w)$ при $1 \leq k < T$.

c) Докажите, что из $v \Rightarrow w$ следует $w \preceq D_k(v)$ для некоторого k, $1 \leq k < T$.

d) Следовательно, схема слияния, сортирующая максимальное число начальных серий на T лентах за q фаз, может быть изображена последовательностью целых чисел $k_1 k_2 \dots k_q$, такой, что начальное распределение есть $D_{k_q}(\dots(D_{k_2}(D_{k_1}(u)))\dots)$, где $u = (1, 0, \dots, 0)$. Эта стратегия минимума фаз имеет сильное T-fifo-представление и входит в класс схем из упр. 22. Когда $T = 3$, это *многофазная* схема, а при $T = 4$, 5, 6, 7 это вариация *сбалансированной* схемы.

26. [*M46*] (Р. М. Карп.) Верно ли, что оптимальная последовательность $k_1 k_2 \dots k_q$, упомянутая в упр. 25, всегда равна $1\lceil T/2 \rceil \lfloor T/2 \rfloor \lceil T/2 \rceil \lfloor T/2 \rfloor \dots$ для всех $T \geq 4$ и всех достаточно больших q?

*5.4.5. Осциллирующая сортировка

Еще один подход к сортировке методом слияния был предложен Шелдоном Собелем (Sheldon Sobel) в *JACM* **9** (1962), 372–375. Вместо того чтобы начинать с распределения прохода, когда все начальные серии распределяются по лентам, он предложил алгоритм, который переключается то на распределение, то на слияние, так что бóльшая часть сортировки происходит еще до того, как вся исходная информация полностью проанализирована.

Предположим, например, что для слияния используются 5 лент. По методу Собеля 16 начальных серий будут сортироваться следующим образом.

	Операция	T1	T2	T3	T4	T5	Стоимость
Фаза 1	Распределение	A_1	A_1	A_1	A_1	—	4
Фаза 2	Слияние	—	—	—	—	D_4	4
Фаза 3	Распределение	—	A_1	A_1	A_1	D_4A_1	4
Фаза 4	Слияние	D_4	—	—	—	D_4	4
Фаза 5	Распределение	D_4A_1	—	A_1	A_1	D_4A_1	4
Фаза 6	Слияние	D_4	D_4	—	—	D_4	4
Фаза 7	Распределение	D_4A_1	D_4A_1	—	A_1	D_4A_1	4
Фаза 8	Слияние	D_4	D_4	D_4	—	D_4	4
Фаза 9	Слияние	—	—	—	A_{16}	—	16

Здесь, как и в разделе 5.4.4, A_r и D_r применяются для обозначения соответственно восходящих и нисходящих серий относительной длины r. При использовании рассматриваемого метода сначала начальные серии по одной записываются на каждую из четырех лент и сливаются (чтение выполняется в обратном направлении) на пятую ленту. Затем распределение возобновляется, на этот раз циклически сдвинутое на одну позицию вправо по отношению к лентам, и в результате второго слияния образуется еще одна серия D_4. Когда этим способом будут сформированы четыре серии D_4, дополнительное слияние приведет к созданию A_{16}. Процесс можно продолжать, создав еще три A_{16}, слив их в D_{64}, и так до тех пор, пока не исчерпаются исходные данные. При этом заранее знать длину исходных данных не нужно.

Если число начальных серий S есть 4^m, то нетрудно заметить, что этот метод обрабатывает каждую запись ровно $m+1$ раз (один раз во время распределения и m раз во время слияния). Если S лежит между 4^{m-1} и 4^m, можно с помощью фиктивных серий увеличить S до 4^m; следовательно, весь процесс сортировки потребует $\lceil \log_4 S \rceil + 1$ проходов по всем данным. Это именно то, что достигается при сбалансированной сортировке на *восьми* лентах. В общем случае осциллирующая сортировка с T рабочими лентами эквивалентна сбалансированному слиянию с $2(T-1)$ лентами, так как при этом выполняется $\lceil \log_{T-1} S \rceil + 1$ проходов по данным. Если S оказывается степенью $T-1$, то это самое лучшее, что можно получить при *любом* методе с T лентами (здесь достигается нижняя оценка из соотношения 5.4.4–(9)). С другой стороны, если S равно $(T-1)^{m-1}+1$, т. е. ровно на единицу больше степени $T-1$, при этом методе теряется почти целый проход.

В упр. 2 показано, как, используя специальную программу завершения, устранить часть этой лишней работы для случая, когда S — не степень T. Еще одно усовершенствование было предложено в 1966 году Деннисом Л. Бенчером (Dennis L. Bencher), который назвал свою процедуру перекрестным слиянием. [См.

H. Wedekind, *Datenorganisation* (Berlin: W. de Gruyter, 1970), 164–166; см. также *U. S. Patent 3540000* (1970).] Основная идея состоит в том, чтобы отложить слияние до тех пор, пока не будет накоплено больше сведений о S. Мы обсудим несколько измененную форму первоначальной схемы Бенчера.

Этот усовершенствованный метод осциллирующей сортировки действует следующим образом.

	Операция	T1	T2	T3	T4	T5	Стоимость
Фаза 1	Распределение	—	A_1	A_1	A_1	A_1	4
Фаза 2	Распределение	—	A_1	A_1A_1	A_1A_1	A_1A_1	3
Фаза 3	Слияние	D_4	—	A_1	A_1	A_1	4
Фаза 4	Распределение	D_4A_1	—	A_1	A_1A_1	A_1A_1	3
Фаза 5	Слияние	D_4	D_4	—	A_1	A_1	4
Фаза 6	Распределение	D_4A_1	D_4A_1	—	A_1	A_1A_1	3
Фаза 7	Слияние	D_4	D_4	D_4	—	A_1	4
Фаза 8	Распределение	D_4A_1	D_4A_1	D_4A_1	—	A_1	3
Фаза 9	Слияние	D_4	D_4	D_4	D_4	—	4

В этот момент слияние всех D_4 в A_{16} не выполняется (если только не окажется, что исходные данные исчерпаны). Лишь после того, как закончится

Фаза 15	Слияние	D_4D_4	D_4D_4	D_4D_4	D_4	—	4,

будет получена первая серия A_{16}

Фаза 16	Слияние	D_4	D_4	D_4	—	A_{16}	16.

Вторая серия A_{16} появится после создания еще трех D_4,

Фаза 22	Слияние	D_4D_4	D_4D_4	D_4	—	$A_{16}D_4$	4
Фаза 23	Слияние	D_4	D_4	—	A_{16}	A_{16}	16,

и т. д. (ср. с фазами 1–5). Преимущества схемы Бенчера можно видеть, если имеется, например, только пять начальных серий: осциллирующая сортировка (ее модификация из упр. 2) выполняла бы четырехпутевое слияние (на фазе 2), за которым следовало бы двухпутевое слияние с общей стоимостью $4 + 4 + 1 + 5 = 14$, тогда как схема Бенчера выполняла бы двухпутевое слияние (на фазе 3), за которым следовало бы четырехпутевое слияние с общей стоимостью $4 + 1 + 2 + 5 = 12$. Оба метода также требуют небольших дополнительных затрат, а именно — однократной перемотки перед окончательным слиянием.

Точное описание метода Бенчера содержится ниже, в алгоритме B. К сожалению, соответствующую процедуру, по-видимому, трудней понять, чем запрограммировать; легче объяснить данный метод компьютеру, чем программисту! Частично это происходит по той причине, что рекурсивный метод выражен в итеративном виде и затем подвергнут некоторой оптимизации; читатель, возможно, обнаружит, что приходится несколько раз проследить за работой алгоритма, прежде чем станет понятно, что же происходит.

Алгоритм B (*Осциллирующая сортировка с "перекрестным" распределением*). Этот алгоритм берет первоначальные серии и распределяет их по лентам, время от времени прерывая процесс распределения, чтобы слить содержимое некоторых

лент (рис. 77). В алгоритме используется P-путевое слияние и предполагается, что имеется $T = P + 1 \geq 3$ накопителей на магнитной ленте (НМЛ) (*не считая* устройства, которое может применяться для хранения исходных данных). НМЛ должны допускать чтение как в прямом, так и в обратном направлениях; они обозначены числами $0, 1, \ldots, P$. Используются следующие массивы.

$\mathtt{D}[j]$, $0 \leq j \leq P$: Число фиктивных серий, наличие которых предполагается в конце j-й ленты

$\mathtt{A}[l, j]$, $0 \leq l \leq L$, $0 \leq j \leq P$: Здесь L — достаточно большое число, такое, что будет введено P^{L+1} начальных серий. Если $\mathtt{A}[l, j] = k \geq 0$, то на ленте j имеется серия номинальной длины P^k, соответствующая "уровню l" работы алгоритма. Эта серия — восходящая, если k четно, и нисходящая, если k нечетно. $\mathtt{A}[l, j] < 0$ означает, что на уровне l лента j не используется

Инструкция "Записать начальную серию на ленту j" является сокращенным обозначением следующих действий.

Установить $\mathtt{A}[l, j] \leftarrow 0$. Если исходные данные исчерпаны, то увеличить $\mathtt{D}[j]$ на 1; в противном случае записать серию на ленту j (в порядке возрастания).

Инструкция "Произвести слияние на ленте j" используется как краткое обозначение следующих действий.

Если $\mathtt{D}[i] > 0$ для всех $i \neq j$, то уменьшить $\mathtt{D}[i]$ на 1 при всех $i \neq j$ и увеличить $\mathtt{D}[j]$ на 1. В противном случае произвести слияние одной серии на ленте j со всех лент $i \neq j$, таких, что $\mathtt{D}[i] = 0$, и уменьшить $\mathtt{D}[i]$ на 1 для всех остальных $i \neq j$.

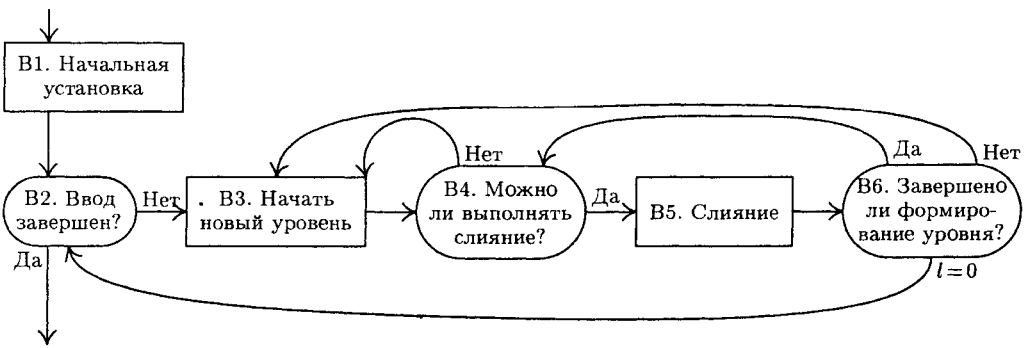


Рис. 77. Осциллирующая сортировка с "перекрестным" распределением.

B1. [Начальная установка.] Установить $\mathtt{D}[j] \leftarrow 0$ при $0 \leq j \leq P$. Установить $\mathtt{A}[0, 0] \leftarrow -1$, $l \leftarrow 0$, $q \leftarrow 0$. Затем записать начальную серию на ленту j для $1 \leq j \leq P$.

B2. [Ввод завершен?] (В этот момент лента q пуста и всякая другая лента содержит максимум одну серию.) Если еще есть исходные данные, перейти к шагу B3. Однако если ввод исчерпан, то перемотать все ленты $j \neq q$, такие, что $\mathtt{A}[0, j]$ четно; затем выполнить слияние на ленту q, читая все только что перемотанные

ленты в прямом направлении, а остальные ленты — в обратном. Этим завершается сортировка; результат находится на ленте q в порядке возрастания.

B3. [Начать новый уровень.] Установить $l \leftarrow l+1$, $r \leftarrow q$, $s \leftarrow 0$ и $q \leftarrow (q+1) \bmod T$. Записать начальные серии на ленты $(q+j) \bmod T$, где $1 \le j \le T-2$. (Таким образом, начальные серии будут записаны на все ленты, кроме лент q и r.) Установить $\mathtt{A}[l,q] \leftarrow -1$ и $\mathtt{A}[l,r] \leftarrow -2$.

B4. [Можно ли выполнять слияние?] Если $\mathtt{A}[l-1,q] \ne s$, вернуться к шагу B3.

B5. [Слияние.] (В этот момент $\mathtt{A}[l-1,q] = \mathtt{A}[l,j] = s$ при всех $j \ne q$, $j \ne r$.) Выполнить слияние на ленте r, читая в обратном направлении. (См. выше определение этой операции.) Затем установить $s \leftarrow s+1$, $l \leftarrow l-1$, $\mathtt{A}[l,r] \leftarrow s$ и $\mathtt{A}[l,q] \leftarrow -1$. Установить $r \leftarrow (2q-r) \bmod T$. (В общем случае имеем $r = (q-1) \bmod T$, если s четно, и $r = (q+1) \bmod T$, если s нечетно.)

B6. [Завершено ли формирование уровня?] Если $l = 0$, перейти к шагу B2. В противном случае, если $\mathtt{A}[l,j] = s$ для всех $j \ne q$ и $j \ne r$, перейти к шагу B4. В противном случае вернуться к B3. ▌

Чтобы показать работоспособность этого алгоритма, можно использовать доказательства наподобие рекурсивной индукции так же, как для алгоритма 2.3.1T. Предположим, что мы начинаем с шага B3 при $l = l_0$, $q = q_0$, $s_+ = \mathtt{A}[l_0, (q_0+1) \bmod T]$ и $s_- = \mathtt{A}[l_0, (q_0-1) \bmod T]$. Допустим, кроме того, что либо $s_+ = 0$, либо $s_- = 1$, либо $s_+ = 2$, либо $s_- = 3$, либо $\cdots$. Можно проверить по индукции, что алгоритм, в конце концов, придет к шагу B5, не изменив строки $\mathtt{A}$ с 0-й по l_0-ю. При этом будут получены следующие значения переменных: $l = l_0+1$, $q = q_0 \pm 1$, $r = q_0$ и $s = s_+$ или s_-. Причем выбирается знак "+", если $s_+ = 0$ или ($s_+ = 2$ и $s_- \ne 1$), или ($s_+ = 4$ и $s_- \ne 1, 3$), или $\cdots$, и знак "−" — если ($s_- = 1$ и $s_+ \ne 0$) или ($s_- = 3$ и $s_+ \ne 0, 2$), или $\cdots$. Приведенный здесь "набросок" доказательства не очень элегантен, но ведь и сам алгоритм сформулирован в виде, который больше годится для реализации, чем для проверки правильности.

На рис. 78 показана эффективность алгоритма B, выраженная средним числом слияний каждой записи в зависимости от S — числа начальных серий, причем предполагается, что начальные серии приблизительно равны по длине. (Соответствующие графики для многофазной и каскадной сортировок приведены на рис. 70 и 74.) При подготовке рис. 78 учтено небольшое усовершенствование, упомянутое в упр. 3. Метод, некоторым образом связанный с рассмотренным и названный *круговой сортировкой* (gyrating sort), разработан Р. М. Карпом (R. M. Karp) и основан на теории слияния в прямом порядке, приведенной в разделе 5.4.4. [См. *Combinatorial Algorithms*, edited by Randall Rustin (Algorithmics Press, 1972), 21–29.]

Прямое чтение. Схема осциллирующей сортировки, по-видимому, требует возможности чтения в обратном порядке, поскольку приходится где-то накапливать длинные серии. Тем не менее в работе M. A. Goetz, *Proc. AFIPS Spring Joint Comp. Conf.* **25** (1964), 599–607, найден способ выполнения осциллирующей сортировки с помощью только прямого чтения и простой перемотки. Предложенный метод в корне отличается от остальных схем, которые мы рассматривали в этой главе, по следующим причинам.

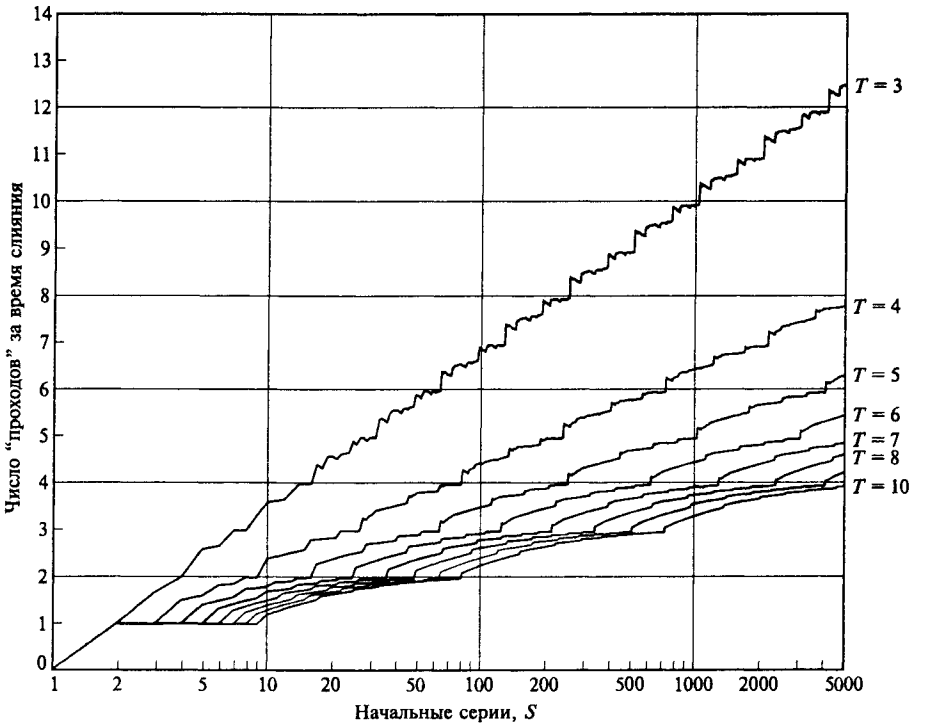


Рис. 78. Эффективность осциллирующей сортировки, использующей метод алгоритма B и упр. 3.

a) Данные иногда записываются в начало ленты, причем предполагается, что данные, которые находятся на *середине* ленты, не разрушаются.

b) Все начальные строки имеют фиксированную максимальную длину.

Условие (a) нарушает свойство "первым включается — первым исключается", которое, как мы предположили, является характеристикой чтения в прямом порядке. Однако оно может быть надежно реализовано, если между сериями оставлять участок чистой ленты достаточной протяженности и если в нужные моменты пренебречь ошибками четности. Условие (b) оказывается до некоторой степени противоречащим эффективному использованию выбора с замещением.

Осциллирующая сортировка Гоетца (Goetz) с прямым чтением имеет одно сомнительное достоинство — это один из первых алгоритмов, который был запатентован как алгоритм, а не как физическое устройство [*U. S. Patent 3380029* (1968)]; если это не удастся оспорить, то из факта патентования следует, что алгоритм нельзя использовать в программе без разрешения владельца патента. Метод Бенчера (осциллирующая сортировка с обратным чтением) был запатентован IBM несколькими годами позже. (Увы, мы являемся свидетелями конца эры, когда удовольствие от открытия нового алгоритма само по себе считалось достаточным вознаграждением! К счастью, осциллирующая сортировка не настолько уж хороша, чтобы из-за нее стоило ломать копья. Будем все-таки надеяться, что здравомыслящие ученые,

придумывая новые алгоритмы, будут продолжать делать свои идеи достоянием всех. Что касается тех недальновидных людей, которые хранят новые алгоритмы в строгом секрете, то нужно сказать, что они немного выигрывают по сравнению с широкой доступностью своих результатов, поскольку право собственности сохраняется в течение лишь ограниченного времени.)

Центральная идея в методе Гоетца состоит в таком использовании лент, чтобы каждая лента начиналась с серии относительной длины 1, за которой следовала бы серия относительной длины P, затем — P^2 и т. д. Например, если $T = 5$, то сортировка начинается следующим образом (“.” указывает текущее положение головки чтения/записи на каждой ленте).

	Операция	T1	T2	T3	T4	T5	“Стоимость”	Примечание
Фаза 1	Распределение	$.A_1$	$.A_1$	$.A_1$	$.A_1$	$A_1.$	5	[T5 не перематывается]
Фаза 2	Слияние	$\not{A}_1.$	$\not{A}_1.$	$\not{A}_1.$	$\not{A}_1.$	$A_1\ A_4.$	4	[Перемотка всех лент]
Фаза 3	Распределение	$.A_1$	$.A_1$	$.A_1$	$A_1.$	$.A_1\ A_4$	4	[T4 не перематывается]
Фаза 4	Слияние	$\not{A}_1.$	$\not{A}_1.$	$\not{A}_1.$	$A_1\ A_4.$	$\not{A}_1.A_4$	4	[Перемотка всех лент]
Фаза 5	Распределение	$.A_1$	$.A_1$	$.A_1$	$.A_1\ A_4$	$.A_1\ A_4$	4	[T3 не перематывается]
Фаза 6	Слияние	$\not{A}_1.$	$\not{A}_1.$	$A_1\ A_4.$	$\not{A}_1.A_4$	$\not{A}_1.A_4$	4	[Перемотка всех лент]
Фаза 7	Распределение	$.A_1$	$A_1.$	$.A_1\ A_4$	$.A_1\ A_4$	$.A_1\ A_4$	4	[T2 не перематывается]
Фаза 8	Слияние	$\not{A}_1.$	$A_1\ A_4.$	$\not{A}_1.A_4$	$\not{A}_1.A_4$	$\not{A}_1.A_4$	4	[Перемотка всех лент]
Фаза 9	Распределение	$A_1.$	$.A_1\ A_4$	$.A_1\ A_4$	$.A_1\ A_4$	$.A_1\ A_4$	4	[T1 не перематывается]
Фаза 10	Слияние	$A_1A_4.$	$\not{A}_1.A_4$	$\not{A}_1.A_4$	$\not{A}_1.A_4$	$\not{A}_1.A_4$	4	[Нет перемотки]
Фаза 11	Слияние	$A_1A_4A_{16}.$	$\not{A}_1\ \not{A}_4.$	$\not{A}_1\ \not{A}_4.$	$\not{A}_1\ \not{A}_4.$	$\not{A}_1\ \not{A}_4.$	16	[Перемотка всех лент]

И так далее. Во время фазы 1 лента T1 перематывается и одновременно на T2 записываются исходные данные; затем перематывается T2 и одновременно на T3 записываются исходные данные и т. д. В конце концов, когда исходные данные исчерпаны, начинают появляться фиктивные серии, и иногда необходимо вообразить, что они записаны явно на ленте полной длины. Например, если $S = 18$, то серии A_1 на T4 и T5 будут зафиксированы во время фазы 9; нам придется продвинуться вперед по T4 и T5 при слиянии с T2 и T3 на T1 во время фазы 10, так как надо добраться до серий A_4 на T4 и T5 для подготовки к фазе 11. С другой стороны, фиктивная серия A_1 на T1 необязательно должна существовать явно. Таким образом, “конец игры” несколько замысловат.

Еще с одним примером применения этого метода мы встретимся в следующем разделе.

УПРАЖНЕНИЯ

1. [*22*] В тексте раздела приводится иллюстрация осциллирующей сортировки Собеля в ее первозданном виде при $T = 5$ и $S = 16$. Дайте точное определение алгоритма, в котором эта процедура обобщается и сортируются $S = P^L$ начальных серий на $T = P+1 \geq 3$ лентах. Постарайтесь найти алгоритм, который может быть очень просто описан.

2. [*24*] Если в изначальном методе Собеля $S = 6$, то мы могли бы заявить, что $S = 16$ и что имеется 11 фиктивных серий. Тогда фаза 3 в примере в тексте раздела поместила бы фиктивные серии A_0 на T4 и T5, фаза 4 слила бы серии A_1 на T2 и T3 в D_2 на T1; фазы 5–8 не делали бы ничего, и фаза 9 породила бы A_6 на T4. Лучше бы перемотать T2 и T3 сразу после фазы 3 и затем немедленно получить A_6 на T4 с помощью трехпутевого слияния.

Покажите, как, основываясь на этой идее, улучшить окончание алгоритма из упр. 1, если S не является точной степенью P.

▶ **3.** [*29*] Составьте таблицу, показывающую поведение алгоритма B при $T = 3$, предполагая, что имеется 9 начальных серий. Покажите, что эта процедура очевидно неэффективна в одном месте, и предложите изменения в алгоритме B, позволяющие исправить этот недостаток.

4. [*21*] На шаге B3 имеется установка отрицательного значения как в $\mathtt{A}[l, q]$, так и в $\mathtt{A}[l, r]$. Покажите, что одна из этих операций всегда лишняя, поскольку соответствующий элемент массива A никогда не рассматривается.

5. [*M25*] Пусть S — число начальных серий в имеющихся исходных данных для алгоритма B. При каких значениях S не требуется *ни одной перемотки* на шаге B2?

*5.4.6. Практическая реализация слияния на лентах

Теперь пришло время заняться "мелочами". В предыдущих разделах было проанализировано множество семейств схем слияния, а в данном будет рассмотрено, как они проявляются в вычислительных системах разных конфигураций при использовании НМЛ различных типов, и выполнено сравнение разных схем слияния с учетом этих характеристик. Анализ внутренней сортировки показал, что невозможно адекватно оценить эффективность того или иного метода, просто подсчитывая число выполняемых при этом сравнений. Аналогично нельзя правильно оценить метод внешней сортировки, ограничившись анализом числа выполняемых проходов по данным.

В этом разделе приводятся характеристики типичных НМЛ и их влияние на начальное распределение и слияние, в частности рассматривается несколько схем распределения буферов и требуемое при их реализации время счета. Здесь также вкратце описывается структура программ — *генераторов сортировки*.

Как работает НМЛ. В характеристиках НМЛ, производимых разными изготовителями, имеются значительные различия. Определим для удобства гипотетический НМЛ MIXT, достаточно типичный для оборудования того времени, когда была написана эта книга. Рассмотрев структуру алгоритма сортировки для НМЛ MIXT, вы лучше поймете, как обращаться с другими конкретными устройствами. MIXT читает или записывает 800 символов на дюйм ленты при скорости протяжки 75 дюймов в секунду. Это означает, что один символ читается или записывается в течение $\frac{1}{60}$ мс, т. е. $16\frac{2}{3}$мкс, если лента находится в активном состоянии. Реальные НМЛ, широко распространенные в 70-х годах, имели плотность записи в диапазоне от 200

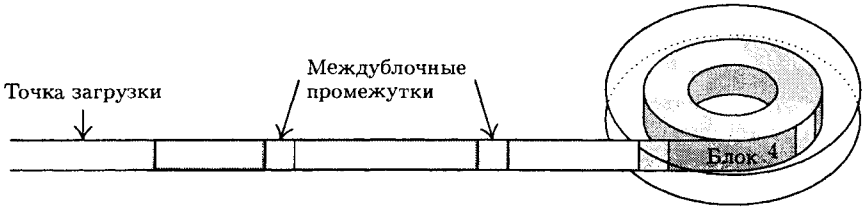


Рис. 79. Магнитная лента с блоками переменной длины.

до 1 600 символов на дюйм и скорость в диапазоне от $37\frac{1}{2}$ до 150 дюймов в секунду, так что их эффективная скорость в сравнении с MIXT изменяется в диапазоне 1/8–4.

Еще в начале раздела 5.4 отмечалось, что, конечно же, в настоящее время НМЛ являются анахронизмом. Но еще в то десятилетие, когда НМЛ были основным компонентом подсистемы внешней памяти, мы подготовили множество примеров и упражнений, сохраняющих актуальность и по сей день. Наша концепция в дальнейшем изложении такова: значение имеет не столько получение конкретного результата, сколько изучение путей приложения теории к конкретному случаю из практики. Я полагаю, что *методология* имеет значительно большее значение, чем *феноменология*, поскольку принципы решения той или иной задачи остаются прежними даже при изменении технологии. Читателям, желающим максимально глубоко проникнуть в суть изложения, я рекомендовал бы мысленно перенестись в середину 70-х годов. *Предположим, что время остановилось и мы живем в прошлом веке компьютерной эры.*

Одно из важных соображений, которое следует иметь в виду, состоит в том, что ленты имеют конечную длину. Каждая бобина содержит 2 400 футов ленты или меньше; следовательно, на одной бобине ленты MIXT есть место для максимум 23 000 000 символов и, чтобы прочесть их все, требуется около 23000000/3600000 $\approx$ 6.4 мин. Если необходимо рассортировать больший файл, то обычно лучше всего рассортировывать за один раз одну полную бобину и затем сливать отдельные рассортированные бобины, чтобы избежать дополнительной работы, связанной с установкой лент. Это означает, что число начальных серий S, реально присутствующих в схемах слияния, которые мы изучали, никогда не будет очень большим. Мы никогда не столкнемся со случаем, когда $S > 5000$, даже при очень маленьком объеме внутренней памяти, что приводит к формированию начальных серий длиной только 5 000 символов. Следовательно, формулы, дающие асимптотическую эффективность алгоритмов при $S \to \infty$, имеют, главным образом, академический интерес.

Данные хранятся на ленте в виде *блоков* (рис. 79), и по каждой команде чтения/записи передается один блок. Блоки часто называются записями, но мы будем избегать этого термина, чтобы не путать его с записями в файле, которые участвуют в сортировке. Для многих ранних программ сортировки, написанных в 50-х годах, это различие было необязательным, так как в одном блоке хранилась одна запись; но мы увидим, что объединение нескольких записей в одном блоке на ленте обычно дает определенные преимущества.

Соседние блоки разделяются *междублочными промежутками* длиной по 480 символов, что позволяет остановить лентопротяжный механизм или разогнать его между отдельными командами чтения или записи. Междублочные промежутки приводят к уменьшению числа символов на одной бобине ленты, зависящему от числа символов в одном блоке (рис. 80); в той же степени уменьшается количество символов, передаваемых за секунду, так как лента движется с постоянной скоростью.

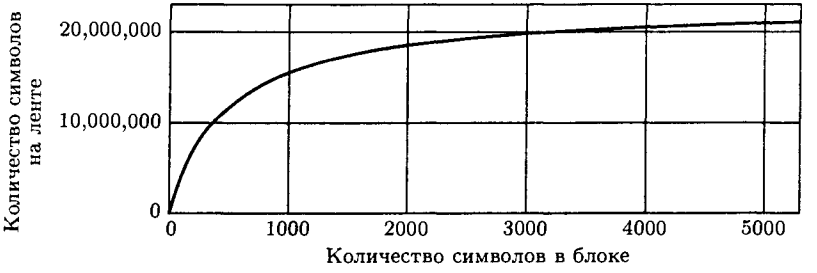


Рис. 80. Число символов на одной бобине ленты MIXT как функция от размера блока.

Многие устаревшие модели компьютеров имеют фиксированные и весьма малые размеры блока данных на магнитной ленте. Например, компьютер MIX, как описано в главе 1, всегда читает и записывает блоки по 100 слов. Таким образом, для MIX получается примерно 500 символов на блок и 480 символов на промежуток, т. е. почти половина ленты пропадает! Сейчас большинство машин допускают работу с переменным размером блока, и поэтому ниже мы обсудим проблему выбора подходящего размера блока.

В конце операции чтения или записи лента проходит с полной скоростью около 66 первых символов междублочного промежутка. Если в это время будет инициирована следующая операция для этой же ленты, то движение ленты продолжится без перерыва. Но если следующая операция не начнется достаточно быстро, то лента остановится и к тому же потребуется некоторое время, чтобы разогнать ее до полной скорости при следующей операции. Суммарная стартстопная задержка составляет 5 мс: 2 — для остановки и 3 — для разгона (рис. 81). Таким образом, если не поддерживать непрерывное движение ленты с полной скоростью, то эффект во времени счета будет, в сущности, такой же, как если бы в междублочном промежутке было 780 символов вместо 480.

Рассмотрим теперь операцию *перемотки*. К сожалению, обычно трудно точно охарактеризовать время, требуемое для перемотки ленты на заданное число символов n. На некоторых машинах имеется быстрая перемотка, которая применяется, только если n превышает число порядка 5 млн; для меньших значений n перемотка происходит с нормальной скоростью чтения/записи. В других НМЛ существует специальный двигатель, используемый во всех операциях перемотки; он постепенно ускоряет бобину ленты до определенного числа оборотов в минуту, а затем тормозит ее, когда подходит время остановки; действительная скорость ленты в этом случае зависит от заполненности бобины. Мы примем для простоты, что MIXT требует $\max(30, n/150)$ мс для перемотки на n символьных позиций (включая промежутки), т. е. примерно две пятых того, что требуется для их чтения/записи. Это достаточно хорошее приближение к поведению многих реальных устройств, где отношение

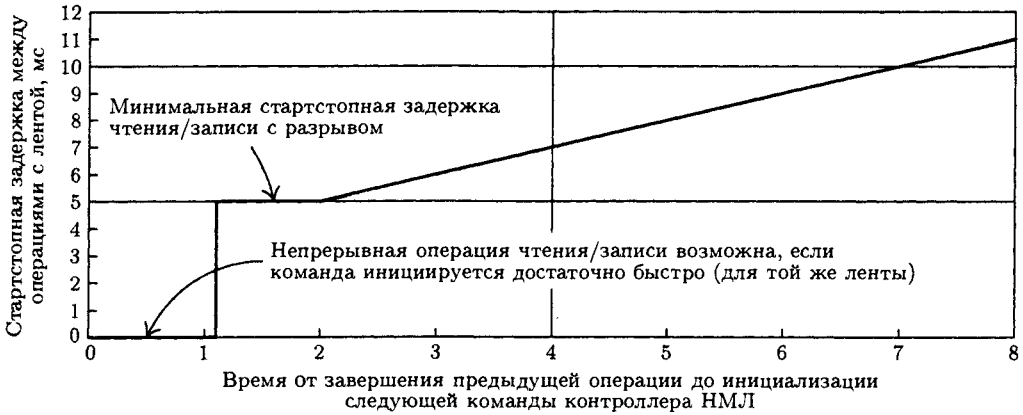


Рис. 81. Вычисление времени стартстопной задержки. (Оно добавляется ко времени, затрачиваемому на чтение или запись блоков и промежутков.)

времени чтения/записи ко времени перемотки обычно находится в диапазоне между 2 и 3, но оно не дает адекватной модели эффекта комбинированной нормальной и быстрой перемотки, имеющейся во многих других моделях НМЛ (рис. 82).

При первоначальной установке и/или перемотке к началу лента помещается в точку загрузки и для любой операции чтения или записи, начинающейся из этого положения, требуется дополнительно 110 мс. Если лента не находится в точке загрузки, она может быть прочитана в обратном направлении; 32 мс добавляется ко времени выполнения любой операции чтения/записи в обратном направлении, если она следует за операцией в прямом направлении, и к любой операции чтения/записи в прямом направлении, следующей за операцией, которая выполняется при обратном направлении движения ленты.

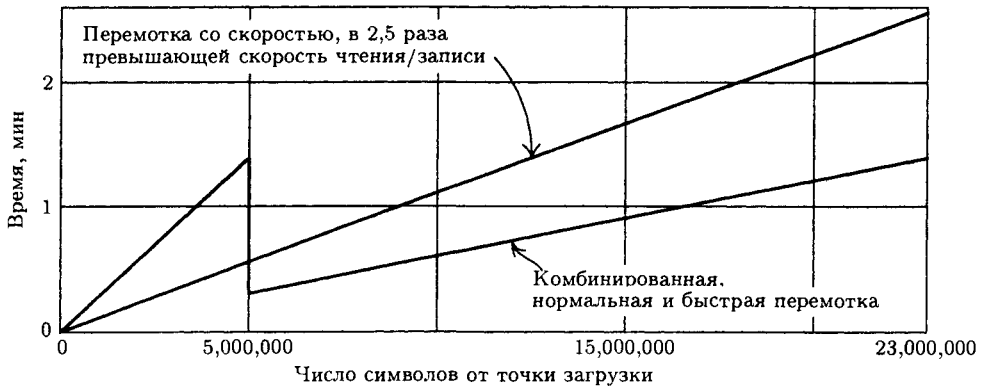


Рис. 82. Приблизительное время счета при двух обычно используемых методах перемотки.

Снова о слиянии. Обратимся вновь к процессу P-путевого слияния, сосредоточив на сей раз внимание на функционировании устройств ввода и вывода. Будем

считать, что для вводных и выводного файлов используется $P + 1$ НМЛ. Наша цель — максимально совместить операции ввода-вывода одну с другой и со счетом по программе так, чтобы минимизировать общее время слияния.

Поучительно рассмотреть следующий частный случай, в котором на возможную степень одновременности наложены серьезные ограничения. Предположим, что

a) в каждый момент времени можно выполнить запись не более чем на одну ленту;

b) в каждый момент времени можно читать не более чем с одной ленты;

c) чтение, запись и вычисления могут происходить одновременно, только если операции чтения и записи были начаты одновременно.

Оказывается, что даже при таких ограничениях достаточно иметь $2P$ буферов ввода и 2 буфера вывода для поддержки, в сущности, максимальной скорости движения лент, если только мы имеем дело не с очень медленным компьютером. Заметим, что (a) не является на самом деле ограничением, так как имеется только одна выводная лента. Кроме того, объем ввода равен объему вывода, так что читается в среднем только одна лента в любой данный момент времени; если условие (b) не выполняется, то обязательно должны быть периоды, когда ввод вообще не происходит. Следовательно, для того чтобы минимизировать время слияния, достаточно поддерживать выводную ленту в рабочем состоянии.

Желаемый эффект дает важный метод, называемый предсказанием или *прогнозированием* (forecasting). Во время выполнения P-путевого слияния обычно имеется P *текущих буферов ввода*, которые используются как источники данных; одни из них заполнены больше других в зависимости от того, какая часть данных в них уже просмотрена. Если все они опустошатся примерно в одно и то же время, то, прежде чем продолжить работу, нам придется выполнить много операций чтения, если только мы не предпримем нужных мер заранее. К счастью, всегда можно сказать, какой буфер первым станет пустым, просто посмотрев на *последнюю* запись в каждом буфере. Буфер, последняя запись которого имеет наименьший ключ, обязательно станет первым пустым буфером независимо от значений каких-либо других ключей, так что мы всегда знаем, какой файл послужит причиной следующей команды ввода. Алгоритм F раскрывает этот принцип в деталях.

Алгоритм F (*Прогнозирование с плавающими буферами*). Этот алгоритм (рис. 83) управляет буферизацией во время P-путевого слияния длинных файлов при $P \geq 2$. Допустим, что вводные ленты и файлы пронумерованы так: $1, 2, \ldots, P$. В алгоритме используются $2P$ буферов ввода, `I[1]`, ..., `I[`$2P$`]`, два буфера вывода, `O[0]` и `O[1]`, и следующие вспомогательные массивы.

`A[`j`]`, $1 \leq j \leq 2P$: 0, если в буфер `I[`j`]` можно вводить данные;
1 — в противном случае

`B[`i`]`, $1 \leq i \leq P$: Буфер, содержащий последний прочитанный блок из файла i

`C[`i`]`, $1 \leq i \leq P$: Буфер, используемый в настоящий момент для ввода из файла i

`L[`i`]`, $1 \leq i \leq P$: Последний ключ, прочитанный из файла i

`S[`j`]`, $1 \leq j \leq 2P$: Буфер, который следует использовать, когда `I[`j`]` опустошится

В описываемом виде алгоритм никогда не остановит работу. Подходящий способ его "выключения" обсуждается ниже.

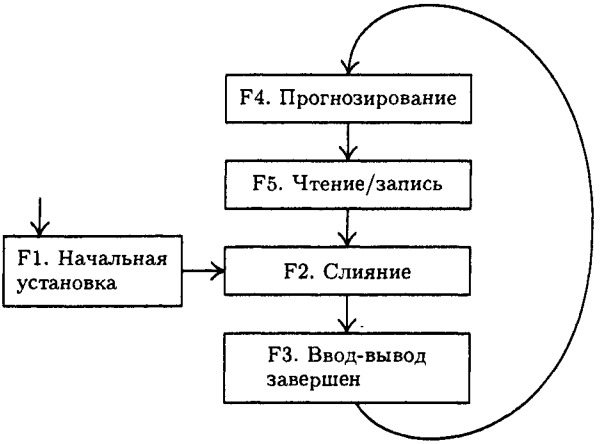


Рис. 83. Прогнозирование с плавающими буферами.

F1. [Начальная установка.] Прочитать первый блок с ленты i в буфер $\mathtt{I}[i]$, установить $\mathtt{A}[i] \leftarrow 1$, $\mathtt{A}[P+i] \leftarrow 0$, $\mathtt{B}[i] \leftarrow i$, $\mathtt{C}[i] \leftarrow i$ и установить $\mathtt{L}[i]$ равным ключу последней записи в $\mathtt{I}[i]$ при $1 \le i \le P$. Затем найти m, такое, что $\mathtt{L}[m] = \min\{\mathtt{L}[1], \ldots, \mathtt{L}[P]\}$, и установить $t \leftarrow 0$, $k \leftarrow P+1$. Начать чтение с ленты m в буфер $\mathtt{I}[k]$.

F2. [Слияние.] Сливать записи из буферов $\mathtt{I}[\mathtt{C}[1]], \ldots, \mathtt{I}[\mathtt{C}[P]]$ в $\mathtt{O}[t]$, пока $\mathtt{O}[t]$ не заполнится. Если во время этого процесса какой-нибудь буфер ввода, скажем $\mathtt{I}[\mathtt{C}[i]]$, станет пустым, а $\mathtt{O}[t]$ еще не будет заполнен, то установить $\mathtt{A}[\mathtt{C}[i]] \leftarrow 0$, $\mathtt{C}[i] \leftarrow \mathtt{S}[\mathtt{C}[i]]$ и продолжать слияние.

F3. [Ввод-вывод завершен.] Ожидать, пока не завершится предыдущая операция чтения (или чтения/записи). Затем установить $\mathtt{A}[k] \leftarrow 1$, $\mathtt{S}[\mathtt{B}[m]] \leftarrow k$, $\mathtt{B}[m] \leftarrow k$ и установить $\mathtt{L}[m]$ равным ключу последней записи в $\mathtt{I}[k]$.

F4. [Прогнозирование.] Найти m, такое, что $\mathtt{L}[m] = \min\{\mathtt{L}[1], \ldots, \mathtt{L}[P]\}$, и найти k, такое, что $\mathtt{A}[k] = 0$.

F5. [Чтение/запись.] Начать чтение с ленты m в буфер $\mathtt{I}[k]$ и выполнить запись из буфера $\mathtt{O}[t]$ на выводную ленту, затем положить $t \leftarrow 1-t$ и вернуться к F2. ▮

В примере на рис. 84 показано, как работает метод прогнозирования при $P = 2$ в предположении, что каждый блок на ленте содержит только две записи. Здесь представлено содержимое буферов ввода во все моменты, когда мы достигаем начала шага F2. Алгоритм F, в сущности, образует P *очередей буферов*, где $\mathtt{C}[i]$ — на начало i-й очереди, $\mathtt{B}[i]$ — на ее конец, $\mathtt{S}[j]$ указывает на преемника буфера $\mathtt{I}[j]$; этим указаниям на рис. 84 соответствуют стрелки. Строка 1 отражает состояние дел после начальной установки. Для каждого вводного файла есть один буфер, и еще один блок читается из файла 1 (так как $03 < 05$). Строка 2 показывает положение вещей после того, как слит первый блок: мы выводим блок, содержащий $\boxed{01\ 02}$, и вводим следующий блок из файла 2 (так как $05 < 09$). Заметим, что в строке 3 три из четырех буферов ввода, по сути дела, предоставлены файлу 2, так как мы выполняем чтение из этого файла и в его очереди уже есть полный буфер и частично заполненный буфер. Этот механизм плавающих буферов является важной чертой

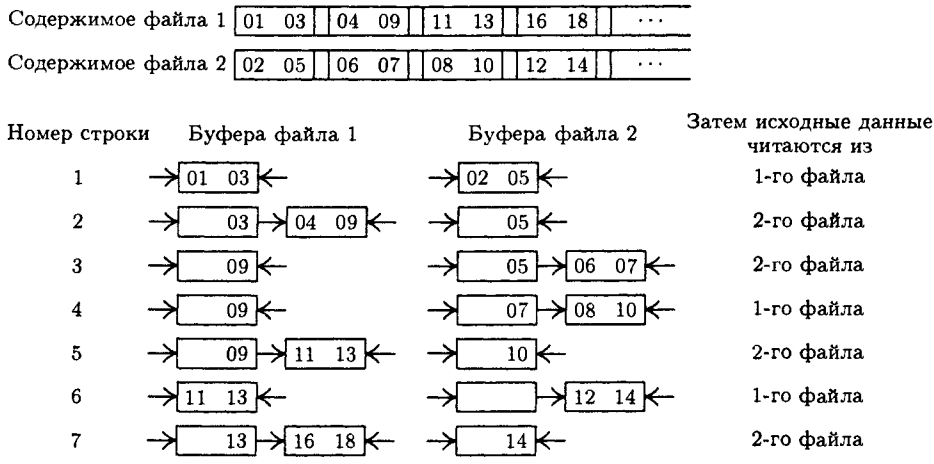


Рис. 84. Очереди буферов в соответствии с алгоритмом F.

алгоритма F, поскольку мы не смогли бы продолжить работу в строке 4, если бы в строке 3 выбрали файл 1 для ввода вместо файла 2.

Чтобы доказать работоспособность алгоритма F, мы должны проверить выполнение двух условий во время реализации алгоритма:

i) всегда имеется свободный буфер (т. е. всегда можно найти k на шаге F4);

ii) если буфер ввода исчерпывается во время слияния, то его преемник уже присутствует в памяти (т. е. `S[C[i]]` на шаге F2 имеет осмысленное значение).

Допустим, что условие (i) не выполняется, т. е. все буфера заняты в некоторый момент, когда мы достигаем шага F4. Каждый раз, когда мы приходим к этому шагу, суммарный объем необработанных данных во всех буферах составляет ровно P емкостей буфера, т. е. данных ровно столько, сколько нужно для заполнения P буферов, ибо данные вводятся и выводятся с одинаковой скоростью. Некоторые буфера заполнены лишь частично, однако для каждого файла частично заполнен максимум один буфер, так что всего таких буферов не более P. По предположению все $2P$ буферов заняты, так что по меньшей мере P из них должны быть заполнены целиком. Это может случиться, только если P буферов полны и P пусты, иначе мы бы имели слишком много данных. Но максимум один буфер может быть одновременно пуст и недоступен; следовательно, условие (i) не может не выполняться.

Допустим, что условие (ii) не выполняется, т. е. для некоторого файла в памяти нет необработанных записей, но текущий буфер вывода еще не полон. Согласно принципу прогнозирования нужно иметь не более одного блока данных для всех остальных файлов, так как мы не читаем блок из файла, если этот блок не потребуется прежде, чем будут исчерпаны буфера какого-нибудь другого файла. Таким образом, общее число необработанных записей составляет максимум $P - 1$ блоков; добавление неполного буфера вывода дает привод к менее чем P буферным емкостям данных в памяти; значит, получено противоречие.

Этот анализ подтверждает работоспособность алгоритма F; он также показывает, что возможны особые ситуации, при которых алгоритм едва-едва избегает

крушения. Нами здесь не упомянута еще одна важная тонкость, касающаяся возможного равенства ключей; это обсуждается в упр. 5. [См. также упр. 4, в котором рассматривается случай $P = 1$.]

Один из способов изящного завершения алгоритма F состоит в присвоении $L[m]$ значения ∞ на шаге F3, если только что прочитанный блок был последним в серии. (Конец серии всегда некоторым особым образом помечается.) После того как будут прочитаны все данные во всех файлах, мы, в конце концов, обнаружим на шаге F4, что все L равны ∞; тогда обычно можно начать чтение первых блоков следующих серий в каждом файле, выполняя начальную установку для следующей фазы слияния по мере вывода последних $P + 1$ блоков.

Таким образом, можно поддерживать полную скорость работы выводной ленты, осуществляя чтение в любое время не более одной ленты. Исключение из этого правила встречается на шаге F1, где было бы полезно читать сразу несколько лент, чтобы обеспечить возможность начать работу; но обычно шаг F1 можно организовать так, чтобы он совмещался с предыдущей частью вычислений.

Идея просмотра последней записи каждого блока, чтобы предсказать, какой буфер первым станет пустым, была высказана в 1953 году Ф. Э. Гольбертон (F. E. Holberton). Сам метод впервые был опубликован в работе E. H. Friend, *JACM* **3** (1956), 144–145, 165. В этом довольно сложном алгоритме использовалось $3P$ буферов ввода и каждому файлу ввода предназначалось по три буфера. Алгоритм F несколько более совершенен, поскольку в нем используются плавающие буфера, что позволяет, таким образом, любому одному файлу потребовать сразу даже $P+1$ буферов ввода, хотя всего требуется не более $2P$ буферов. Слияние с менее чем $2P$ буферами обсуждается в конце этого раздела. Некоторые интересные подходы к совершенствованию алгоритма F рассматриваются в разделе 5.4.9.

Сравнительный анализ схем слияния. Используем теперь наши знания о лентах и слиянии, чтобы сравнить эффективность различных схем слияния, рассмотренных в разделах 5.4.2–5.4.5. Будет весьма поучительно проанализировать детально каждый метод в применении к одному и тому же примеру. Рассмотрим поэтому сортировку файла, каждая запись которого содержит 100 символов, причем в памяти для записи данных доступно 100,000 символьных ячеек (не считая места для программы, ее вспомогательных переменных и сравнительно небольшого пространства, необходимого для ссылок в дереве выбора). Исходные данные расположены на ленте в случайном порядке блоками по 5 000 символов, и результат должен получиться в том же формате. В нашем распоряжении имеется пять рабочих НМЛ в дополнение к устройству, на котором находится вводная лента.

Общее число сортируемых записей — 100,000, но эта информация алгоритму сортировки заранее не известна.

В диаграмму A (здесь и далее см. вкладку) сведены те операции, которые выполняются, когда к нашим данным применяется десять различных схем слияния. Обратившись к этой важной иллюстрации, очень полезно вообразить, что вы наблюдаете за реальной сортировкой: медленно просматривайте каждую строку слева направо, мысленно представляя себе шесть лент, осуществляющих чтение, запись, перемотку и/или обратное чтение, как указано на диаграмме. В течение P-путевого слияния вводные ленты будут находиться в движении в $1/P$ раз реже, чем выводная лента. Заметим, что на диаграмме A предполагается, что, когда пер-

воначальная вводная лента полностью прочитана (и перемотана, чтобы ее убрать), умелый оператор снимает ее и заменяет рабочей лентой за 30 с. В примерах 2–4 это и есть время критического пути, когда компьютер в бездействии ожидает оператора. Но в остальных примерах операция снятия и установки лент совмещена с другой работой. (На диаграмме А по горизонтали указано время в минутах.)

Пример 1. Сбалансированное слияние с прямым чтением. Напомним описание задачи: записи имеют длину 100 символов, внутренней памяти достаточно для одновременного хранения 1 000 записей и каждый блок вводной ленты содержит 5 000 символов (50 записей). Всего имеется 100,000 записей (что равно 10,000,000 символов или 2 000 блоков).

Мы ничем не связаны в выборе размера блоков для промежуточных файлов. При шестиленточном сбалансированном слиянии используется трехпутевое слияние, так что техника алгоритма F требует 8 буферов; можно, следовательно, использовать блоки, содержащие по $1000/8 = 125$ записей (или 12 500 символов).

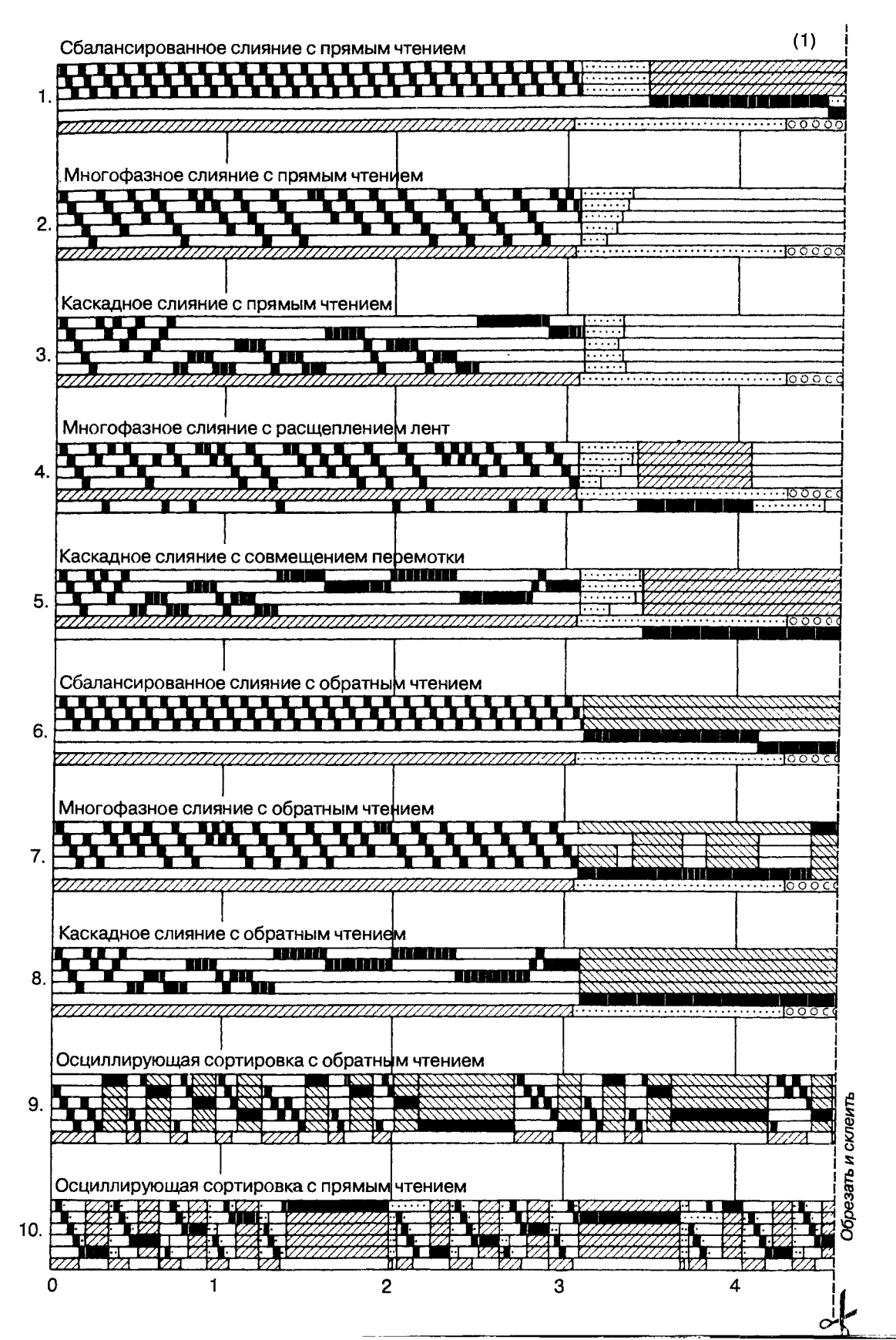
Проход начального распределения может использовать выбор с замещением (алгоритм 5.4.1R), и, чтобы поддерживать непрерывную работу лент, будем использовать два буфера ввода по 50 записей плюс два буфера вывода по 125 записей. Это оставляет для дерева выбора место объемом 650 записей. Бóльшая часть начальных серий будет, следовательно, иметь длину около 1 300 записей (10 или 11 блоков); на диаграмме А получилось 78 начальных серий, причем последняя серия короткая.

При первом проходе слияния, как показано, сливаются 9 серий на ленту 4, а не чередуются ленты 4–6. Это дает возможность выполнять полезную работу в то время, когда оператор вычислительной машины устанавливает рабочую ленту на устройство 6; так как общее число серий S известно сразу после завершения начального распределения, то алгоритм знает, что на ленту 4 должно быть слито $\lceil S/9 \rceil$ серий, затем $\lceil (S-3)/9 \rceil$ — на ленту 5, затем $\lceil (S-6)/9 \rceil$ — на ленту 6.

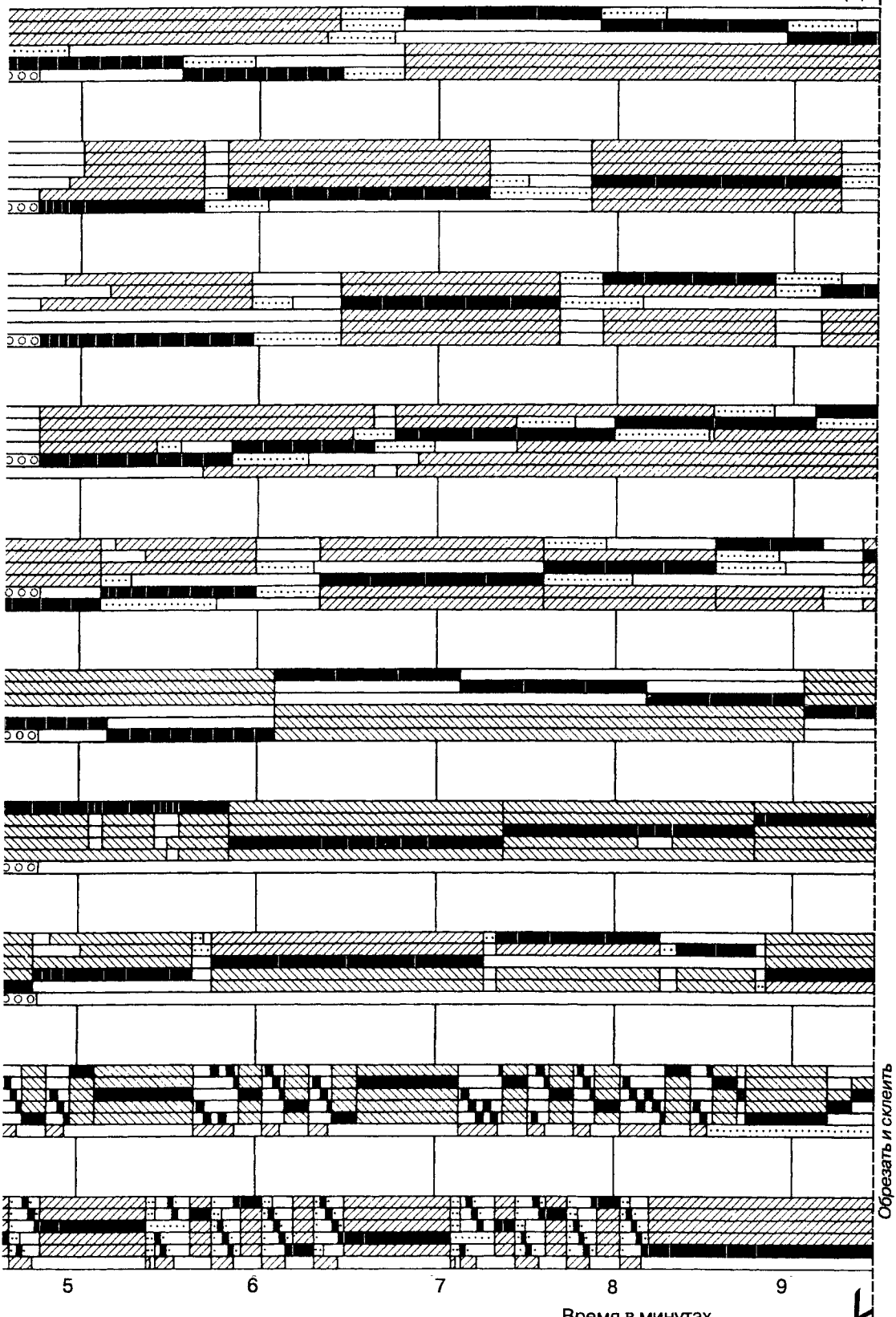
Вся процедура сортировки в этом примере может быть изображена с использованием обозначений, введенных в разделе 5.4.2, следующим образом:

1^{26}	1^{26}	1^{26}	—	—	—
—	—	—	3^9	3^9	3^8
9^3	9^3	$9^2 6^1$	—	—	—
—	—	—	27^1	27^1	24^1
78^1	—	—	—	—	—

Пример 2. Многофазное слияние с прямым чтением. Второй пример на диаграмме А иллюстрирует многофазное слияние в соответствии с алгоритмом 5.4.2D. В этом случае мы выполняем пятипутевое слияние, поэтому память разбита на 12 буферов по 83 записи. В течение первоначального выбора с замещением мы имеем два буфера ввода объемом 50 записей и два буфера вывода объемом 83 записи, что оставляет 734 записи в дереве; таким образом, начальные серии на этот раз будут иметь длину около 1 468 записей (17 или 18 блоков). В данной ситуации получено $S = 70$ начальных серий, причем длины двух последних в действительности равны только четырем блокам и одному блоку соответственно. Схему слияния можно изобразить так:


(1)
Сбалансированное слияние с прямым чтением
1.
Многофазное слияние с прямым чтением
2.
Каскадное слияние с прямым чтением
3.
Многофазное слияние с расщеплением лент
4.
Каскадное слияние с совмещением перемотки
5.
Сбалансированное слияние с обратным чтением
6.
Многофазное слияние с обратным чтением
7.
Каскадное слияние с обратным чтением
8.
Осциллирующая сортировка с обратным чтением
9.
Осциллирующая сортировка с прямым чтением
10.
0
1
2
3
4
Обрезать и склеить

(2)

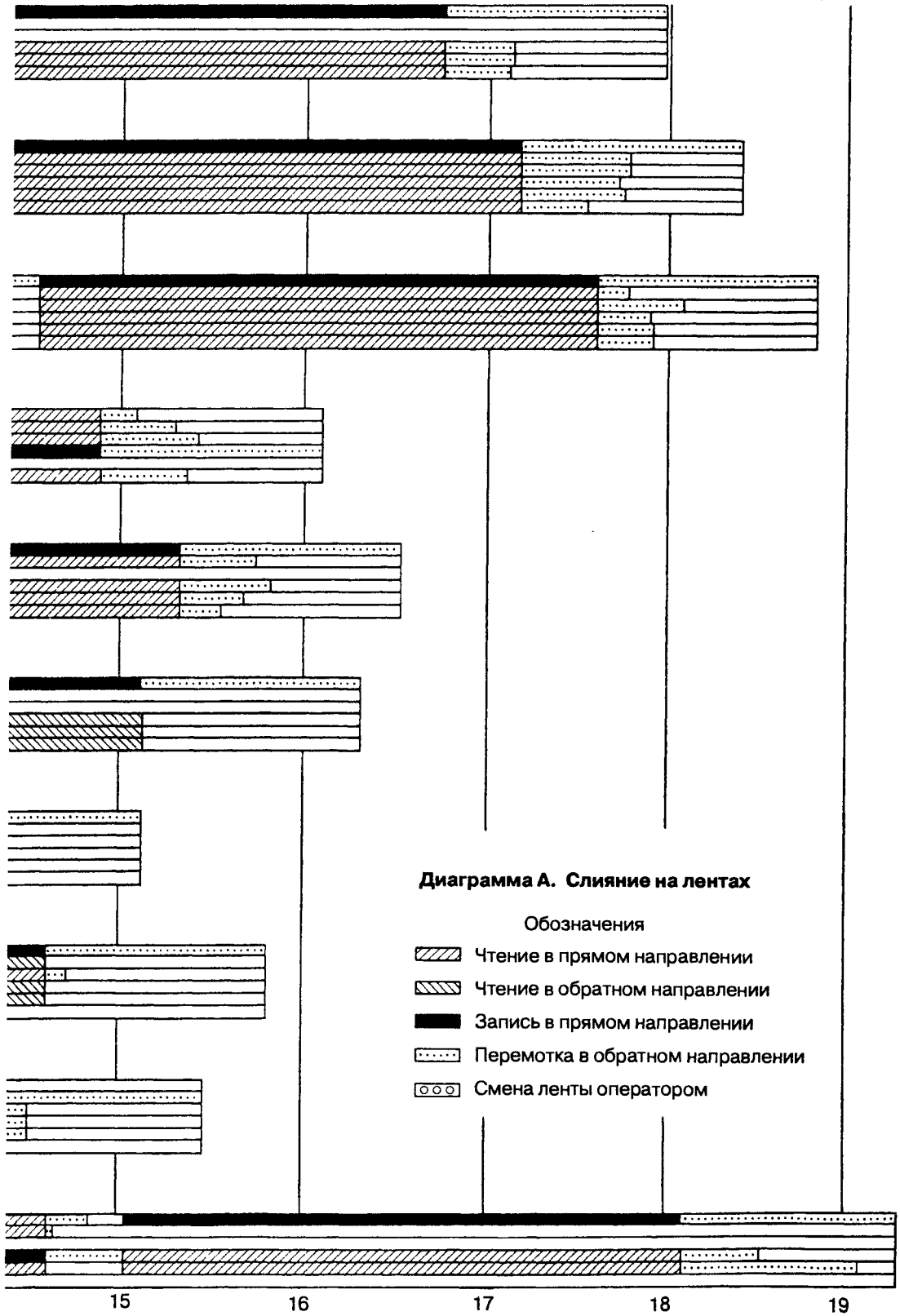


Вырезать из книги

Обрезать и склеить

Вырезать из книги

Обрезать и склеить



Вырезать из книги

$0^{13}1^{18}$	$0^{13}1^{17}$	$0^{13}1^{15}$	$0^{12}1^{12}$	$0^{8}1^{8}$	—
1^{15}	1^{14}	1^{12}	1^{8}	—	$0^{8}1^{4}2^{1}5^{3}$
1^{7}	1^{6}	1^{4}	—	4^{8}	$1^{4}2^{1}5^{3}$
1^{3}	1^{2}	—	8^{4}	4^{4}	$2^{1}5^{3}$
1^{1}	—	$16^{1}19^{1}$	8^{2}	4^{2}	5^{2}
—	34^{1}	19^{1}	8^{1}	4^{1}	5^{1}
70^{1}	—	—	—	—	—

Удивительно, но для многофазного слияния необходимо на 25 с *больше*, чем для значительно более простого сбалансированного слияния! Это объясняется двумя основными причинами.

1) Этот случай особенно удачен для сбалансированного слияния, так как $S = 78$ очень близко к точной степени 3. Если бы было получено 82 начальные серии, то сбалансированное слияние потребовало бы еще одного прохода.

2) При многофазном слиянии теряется 30 с во время замены вводной ленты и в целом свыше 5 мин проходит в ожидании завершения операций перемотки. В противоположность этому сбалансированное слияние требовало сравнительно небольшого времени перемотки. Во второй фазе многофазного слияния сэкономлено 13 с, так как 8 фиктивных серий на ленте 6 можно считать присутствующими даже во время перемотки этой ленты, но затем не происходит никакого совмещения перемотки. Таким образом, многофазный метод проигрывает, несмотря на то что он требует значительно меньшего времени чтения/записи.

Пример 3. Каскадное слияние с прямым чтением. Этот случай аналогичен предыдущему, только в нем используется алгоритм 5.4.3C:

1^{14}	1^{15}	1^{12}	1^{14}	1^{15}	—
1^{5}	1^{9}	—	1^{14}	1^{15}	$1^{3}2^{3}3^{6}$
$5^{1}6^{3}$	5^{3}	$5^{3}6^{2}$	—	1^{1}	2^{2}
—	12^{1}	6^{1}	18^{1}	18^{1}	16^{1}
70^{1}	—	—	—	—	—

(Просматривая диаграмму A, не забывайте представлять каждый пример в действии.)

Пример 4. Многофазное слияние с расщеплением лент. Эта процедура, описанная в конце раздела 5.4.2, позволяет совместить бóльшую часть времени перемотки. Она использует четырехпутевое слияние, так что мы делим память на 10 буферов по 100 записей; в дереве выбора имеется 700 записей, и в результате оказывается, что образовано 72 начальные серии. Последняя серия вновь очень короткая. Использована схема распределения, аналогичная представленной в алгоритме 5.4.2D, за ней следует простой, но в некоторой степени специализированный метод размещения фиктивных серий:

1^{21}	1^{19}	1^{15}	1^{8}	—	$0^{2}1^{9}$
$0^{2}1^{17}$	$0^{2}1^{15}$	$0^{2}1^{11}$	$0^{2}1^{4}$	—	$0^{2}1^{9}4^{4}$

1^{13}	1^{11}	1^{7}	—	$0^2 4^4$	$0^2 1^9 4^4$
1^{10}	1^{8}	1^{4}	—	$0^2 4^4 3^2 4^1$	$1^8 4^4$
1^{6}	1^{4}	—	4^4	$0^2 4^4 3^2 4^1$	$1^4 4^4$
1^{5}	1^{3}	—	$4^4 3^1$	$0^1 4^4 3^2 4^1$	$1^3 4^4$
1^{2}	—	$3^1 7^2$	$4^4 3^1$	$4^2 3^2 4^1$	4^4
1^{1}	—	$3^1 7^2 13^1$	$4^3 3^1$	$4^1 3^2 4^1$	4^3
—	13^1	$3^1 7^2 13^1$	$4^2 3^1$	$3^2 4^1$	4^2
—	$13^1 14^1$	$7^2 13^1$	$4^1 3^1$	$3^1 4^1$	4^1
18^1	$13^1 14^1$	$7^1 13^1$	3^1	4^1	—
18^1	14^1	13^1	—	—	27^1
—	—	—	72^1	—	—

Среди всех примеров на диаграмме А, в которых не используется чтение в обратном направлении, в этом, как оказывается, наилучшее время выполнения. Поскольку S никогда не бывает очень большим, можно разработать более сложный алгоритм, который позволит разместить фиктивные серии еще лучше (см. упр. 5.4.2–(26)).

Пример 5. Каскадное слияние с совмещением перемоток. Эта процедура работает почти так же быстро, как предыдущая, хотя управляющий ею алгоритм более прост. Мы используем для начального распределения метод каскадной сортировки, как в алгоритме 5.4.3С, но с $T = 5$, а не $T = 6$. Затем использование лент в каждой фазе каждого "каскада" чередуется таким образом, что мы обычно не выполняем запись на ленту, пока она почти наверняка не окажется перемотанной. Короче говоря, схема такова:

1^{21}	1^{22}	1^{19}	1^{10}	—	—
1^{4}	1^{7}	—	—	$1^2 2^2 3^5$	4^{10}
7^{2}	—	8^3	$7^2 8^2$	—	4^1
—	26^1	—	8^1	22^1	16^1
72^1	—	—	—	—	—

Пример 6. Сбалансированное слияние с обратным чтением. Этот пример похож на пример 1, но все перемотки устранены:

A_1^{26}	A_1^{26}	A_1^{26}	—	—	—
—	—	—	D_3^9	D_3^9	D_3^8
A_9^3	A_9^3	$A_9^2 A_6^1$	—	—	—
—	—	—	D_{24}^1	D_{27}^1	D_{27}^1
A_{78}^1	—	—	—	—	—

Так как в примере 1 было сравнительно мало перемоток, эта схема не намного лучше, чем в случае прямого чтения. Фактически она оказывается несколько медленнее многофазной схемы с расщеплением лент, несмотря на удачное значение $S = 78$.

Пример 7. Многофазное слияние с обратным чтением. В этом примере используется только пять лент из шести, чтобы устранить время перемотки и смены вводной ленты. Таким образом, применяется только четырехпутевое слияние и такая же структура буферов, как в примерах 4 и 5. Используется распределение, аналогичное приведенному в алгоритме 5.4.2D, но направление серий чередуется и лента 1 фиксируется, как конечная выводная лента. Сначала записывается восходящая серия на ленту 1, затем — нисходящие серии 2–4, после этого — восходящие серии на ленты 2–4, нисходящие серии на ленты 1–3 и т. д. Всякий раз, когда переключается направление, выбор с замещением обычно дает более короткую серию, поэтому было образовано 77 начальных серий вместо 72 в примерах 4 и 5.

Эта процедура в результате дает распределение (22, 21, 19, 15) серий, а ближайшее точное распределение — (29, 56, 52, 44). В упр. 5.4.4–5 показано, как построить строки чисел слияния, которые могут использоваться для размещения фиктивных серий оптимальным образом. Такая процедура может выполняться на практике, поскольку конечность бобины гарантирует, что S никогда не будет слишком большим. Поэтому пример на диаграмме А был построен с использованием такого метода размещения фиктивных серий (см. упр. 7). Он оказался самым быстрым из всех представленных примеров.

Пример 8. Каскадное слияние с обратным чтением. Как и в примере 7, здесь участвует только 5 лент. Эта процедура соответствует алгоритму 5.4.3C, используя перемотку и прямое чтение, чтобы избежать однопутевого слияния (так как перемотка более чем в два раза быстрее чтения на устройствах MIXT). Распределение, следовательно, то же, что и в примере 6. Используя символ "↓" для обозначения перемотки, изобразим эту схему так:

A_1^{21}	A_1^{22}	A_1^{19}	A_1^{10}	—
$A_1^4\downarrow$	$A_1^7\downarrow$	—	$D_1^2D_2^2D_3^5$	D_4^{10}
$A_8A_7^2$	A_5^2	A_9^4	—	$D_4^1\downarrow$
—	D_{17}	$A_9\downarrow$	D_{25}	D_{21}
A_{72}	—	—	—	—

Пример 9. Осциллирующая сортировка с обратным чтением. При осциллирующей сортировке с $T = 5$ (алгоритм 5.4.5B) может использоваться распределение буферов, как в примерах 4, 5, 7 и 8, поскольку она выполняет четырехпутевое слияние. Однако выбор с замещением действует здесь иначе, так как непосредственно перед входом в каждую фазу слияния выводится серия длиной 700 (а не примерно 1 400), чтобы очистить внутреннюю память. Следовательно, здесь порождается 85 серий вместо 72. Некоторые ключевые шаги этого процесса таковы:

—	A_1	A_1A_1	A_1A_1	A_1A_1
D_4	—	A_1	A_1	A_1
. . .	. . .	. . .	. . .	. . .
D_4D_4	D_4D_4	D_4D_4	D_4	—
D_4	D_4	D_4	—	A_{16}

. . .	. . .	. . .	. . .	. . .
D_4	$A_{16}D_4D_4$	$A_{16}D_4$	$A_{16}D_4A_1$	A_{16}
D_4	$A_{16}D_4D_4$	$A_{16}D_4D_1$	$A_{16}D_4$	A_{16}
—	$A_{16}D_4$	$A_{16}D_4$	A_{16}	$A_{16}A_{13}$
—	$A_{16}D_4$	A_{16}	$A_{16}A_4$	$A_{16}A_{13}$
—	A_{16}	$A_{16}A_4$	$A_{16}A_4$	$A_{16}A_{13}$
D_{37}	—	$A_{16}\downarrow$	$A_{16}\downarrow$	$A_{16}\downarrow$
—	A_{85}	—	—	—

Пример 10. Осциллирующая сортировка с прямым чтением. В последнем примере выбор с замещением не используется, так как все начальные серии должны быть одной длины. Следовательно, будет происходить внутренняя сортировка 1 000 записей (полной емкости памяти) каждый раз, когда требуется начальная серия; это дает $S = 100$. Вот некоторые ключевые шаги процесса:

A_1	A_1	A_1	A_1	A_1
—	—	—	—	A_1A_4
. . .	. . .	. . .	. . .	. . .
A_1	A_1	A_1	A_1	A_1A_4
—	—	—	A_1A_4	$\cancel{A_1}A_4$
—	—	—	A_1A_4	A_1A_4
. . .	. . .	. . .	. . .	. . .
A_1	A_1A_4	A_1A_4	A_1A_4	A_1A_4
A_1A_4	$\cancel{A_1}A_4$	$\cancel{A_1}A_4$	$\cancel{A_1}A_4$	$\cancel{A_1}A_4$
$A_1A_4A_{16}$	—	—	—	—
. . .	. . .	. . .	. . .	. . .
—	A_1A_4	A_1A_4	A_1A_4	$A_1A_4A_{16}A_{64}$
A_4	$\cancel{A_1}A_4$	$\cancel{A_1}A_4$	$\cancel{A_1}A_4$	$\cancel{A_1}A_4A_{16}A_{64}$
A_4A_{16}	—	—	—	$\cancel{A_1}\cancel{A_4}A_{16}A_{64}$
$\cancel{A_4}A_{16}$	A_4	—	—	$\cancel{A_1}\cancel{A_4}A_{16}A_{64}$
—	—	—	A_{36}	$\cancel{A_1}\cancel{A_4}\cancel{A_{16}}A_{64}$
A_{100}	—	—	—	—

Эта программа оказывается самой медленной из всех частично из-за того, что в ней не используется выбор с замещением, но большей частью вследствие ее весьма нескладного конца (двухпутевое слияние).

Оценка времени выполнения. Посмотрим теперь, как вычислить приблизительное время выполнения некоторого метода сортировки с учетом характеристик используемого НМЛ `MIXT`. Можно ли предсказать результаты, изображенные на диаграмме А, не выполняя детального моделирования?

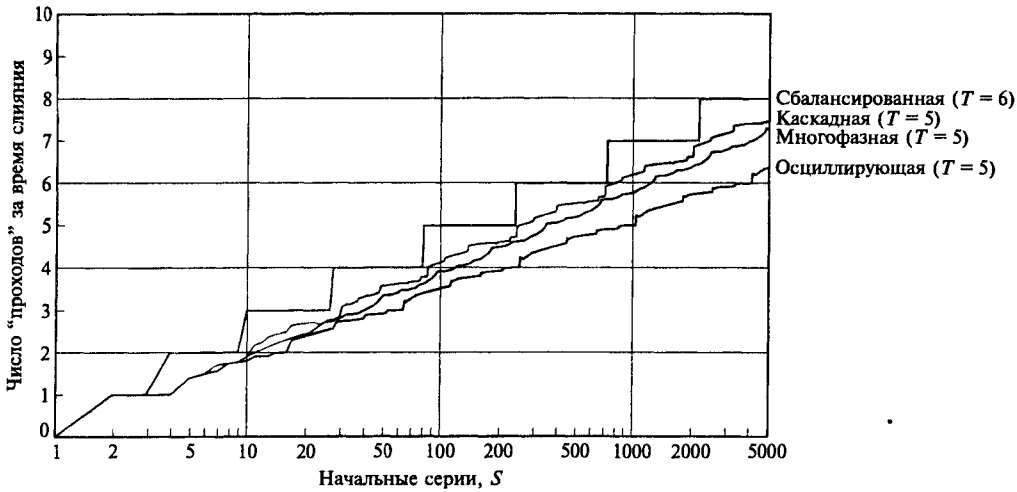


Рис. 85. Несколько обманчивый способ сравнения схем слияния.

Один способ, который традиционно использовался для сравнения различных схем слияния, состоит в том, чтобы совместить графики, подобные представленным на рис. 70, 74 и 78. На этих графиках изображено эффективное число проходов по данным как функция от числа начальных серий в предположении, что все начальные серии имеют примерно равную длину (рис. 85). Но это *не* позволяет выполнить адекватное сравнение, потому что, как мы видели, разные методы приводят к различному числу начальных серий; кроме того, имеются различные накладные расходы, зависящие от относительной частоты междублочных промежутков; значительное воздействие оказывает также время перемотки. Все эти зависящие от конкретного оборудования особенности затрудняют использование машинно-независимой методики истинного сравнения методов слияния. С другой стороны, из рис. 85 все же явствует, что за исключением сбалансированного слияния эффективное число проходов может быть достаточно хорошо аппроксимировано плавными кривыми вида $\alpha \ln S + \beta$. Следовательно, можно неплохо сравнивать методы в любой практической ситуации, проанализировав формулы, аппроксимирующие время выполнения. Конечно, наша цель — найти формулы простые, но достаточно близкие к реальности.

Попытаемся теперь вывести такие формулы в терминах следующих параметров:

N — число сортируемых записей,

C — число символов в записи,

M — объем доступного пространства в оперативной памяти (символов; предполагается кратным C)

τ — время в секундах, нужное для того, чтобы прочитать или записать один символ,

$\rho\tau$ — время в секундах для перемотки в пересчете на один символ,

$\sigma\tau$ — время в секундах стартстопной задержки,

γ — число символов в междублочном промежутке,

δ — время в секундах, необходимое оператору для снятия и замены вводной ленты,

B_i — число символов в блоке нерассортированного ввода,

B_o — число символов в блоке рассортированного вывода.

Для MIXT имеем $\tau = 1/60000$, $\rho = 2/5$, $\sigma = 300$, $\gamma = 480$. В примере, рассмотренном выше, $N = 100000$, $C = 100$, $M = 100000$, $\delta = 30$, $B_i = B_o = 5000$. Обычно именно эти характеристики оборудования и данных решающим образом влияют на время сортировки (хотя время перемотки часто задается более сложным выражением, чем просто коэффициентом ρ). Имея указанные параметры и схему слияния, вычислим еще некоторые величины:

P — максимальный порядок слияния в схеме,

P' — число записей в дереве выбора с замещением,

S — число начальных серий,

$\pi = \alpha \ln S + \beta$ — приблизительное среднее число чтений и записей каждого символа, не считая начального распределения и окончательного слияния,

$\pi' = \alpha' \ln S + \beta'$ — приблизительное среднее число перемоток каждого символа во время промежуточных фаз слияния,

B — число символов в блоке в промежуточных фазах слияния,

$\omega_i, \omega, \omega_o$ — "коэффициенты накладных расходов" — действительное время, необходимое для чтения или записи в пересчете на один символ (с учетом промежутков и стартстопного времени), отнесенное к общему времени τ.

В примерах диаграммы А размеры блоков и буферов выбраны в соответствии с формулой

$$B = \left\lfloor \frac{M}{C(2P+2)} \right\rfloor C, \qquad (1)$$

чтобы блоки могли быть максимально большими при условии совместимости со схемой буферизации алгоритма F. (Во избежание ненужных забот во время последнего прохода величина P должна быть достаточно малой, чтобы (1) обеспечило $B \geq B_o$.) Размер дерева во время выбора с замещением будет, следовательно, таковым:

$$P' = (M - 2B_i - 2B)/C. \qquad (2)$$

Для случайных данных число начальных серий можно оценить формулой

$$S \approx \left\lceil \frac{N}{2P'} + \frac{7}{6} \right\rceil, \qquad (3)$$

опираясь на результаты раздела 5.4.1. Предполагая, что $B_i < B$ и что вводная лента во время распределения может работать с полной скоростью (см. ниже), распределение начальных серий займет примерно $NC\omega_i\tau$ с, где

$$\omega_i = (B_i + \gamma)/B_i. \qquad (4)$$

Во время слияния схема буферизации допускает совмещение чтения, записи и вычислений, но при частом переключении вводных лент необходимо учитывать и стартстопное время; поэтому положим, что

$$\omega = (B + \gamma + \sigma)/B, \tag{5}$$

и время слияния оценим формулой

$$(\pi + \rho\pi')NC\omega\tau. \tag{6}$$

В этой формуле слегка завышена оценка времени перемотки, так как в нее включено стартстопное время, но другие факторы (такие, как взаимная блокировка перемоток и потери времени на чтение с точки загрузки) обычно компенсируют это. Окончательный проход слияния в предположении, что $B_o \leq B$, ограничивается коэффициентом накладных расходов

$$\omega_o = (B_o + \gamma)/B_o. \tag{7}$$

Можно оценить время выполнения последнего слияния и перемотки как

$$nC(1 + \rho)\omega_o\tau;$$

на практике оно могло бы быть несколько больше из-за наличия неравных блоков (ввод и вывод не синхронизированы, как в алгоритме F), но это время будет, в основном, одинаково для всех схем слияния.

Прежде чем переходить к более специфическим формулам для отдельных схем, попробуем обосновать два из сделанных выше предположений.

a) *Может ли система, реализующая выбор с замещением "поспеть" за вводной лентой?* В примере на диаграмме A, вероятно, может, так как для выбора новой записи требуется около десяти итераций внутреннего цикла алгоритма 5.4.1R и в нашем распоряжении время $C\omega_i\tau > 1667$ мкс, за которое следует выполнить эти итерации. Тщательно запрограммировав цикл выбора с замещением, можно достичь этого на многих компьютерах (что и происходило даже в 70-х годах). Заметим, что при слиянии положение несколько менее критическое: время вычисления для одной записи почти всегда меньше времени работы НМЛ при P-путевом слиянии, так как P не очень велико.

b) *Должны ли мы на самом деле выбирать в качестве B максимально возможный размер буфера, как задано в формуле* (1)? Выбрав большой размер буфера, можно сократить отношение издержек ω в (5), но при этом увеличится число начальных серий S, так как P' уменьшается. Сразу и не скажешь, какой фактор важнее. Рассматривая время слияния как функцию от $x = CP'$, можно выразить его в виде

$$\left(\theta_1 \ln\left(\frac{N}{x} + \frac{7}{6}\right) + \theta_2\right)\left(\frac{\theta_3 - x}{\theta_4 - x}\right) \tag{8}$$

для некоторых подходящих констант θ_1, θ_2, θ_3, θ_4, причем $\theta_3 > \theta_4$. Дифференцирование по x показывает, что есть некоторое N_0, такое, что для всех $N \geq N_0$ невыгодно увеличивать x за счет размера буфера. В примерах, приведенных на диаграмме A, N_0 оказалось, грубо говоря, равным 10 000; при сортировке более 10 000 записей большой размер буфера предпочтительнее.

Заметим, однако, что при сбалансированном слиянии число проходов резко изменяется, когда S "проходит" через степень P. Если заранее известно приближенное значение N, то размер буфера следует выбрать таким, чтобы S с большой вероятностью оказалось немного меньше степени P. Например, размер буфера для первого графика диаграммы А был равен 12 500, так как $S = 78$. Это было вполне удовлетворительно, но если бы S оказалось равным 82, было бы значительно лучше немного уменьшить размер буфера.

Формулы для десяти примеров. Возвращаясь к диаграмме А, попытаемся вывести формулы, аппроксимирующие время работы для каждого из десяти методов. В большинстве случаев основная формула

$$NC\omega_i\tau + (\pi + \rho\pi')NC\omega\tau + (1+\rho)NC\omega_o\tau \tag{9}$$

будет достаточно хорошим приближением к суммарному времени сортировки, если мы определим число промежуточных проходов слияния $\pi = \alpha \ln S + \beta$ и число промежуточных проходов перемотки $\pi' = \alpha' \ln S + \beta'$. Иногда необходимо внести в (9) некоторую поправку; специфика каждого метода учитывается следующим образом.

Пример 1. Сбалансированное слияние с прямым чтением. Формулы

$$\pi = \lceil \ln S/\ln P \rceil - 1, \qquad \pi' = \lceil \ln S/\ln P \rceil / P$$

могут использоваться для P-путевого слияния с $2P$ лентами.

Пример 2. Многофазное слияние с прямым чтением. Можно положить $\pi' \approx \pi$, так как за каждой фазой обычно следует перемотка приблизительно такой же длины, как предшествующее слияние. Из табл. 5.4.2–1 получаем в случае шести лент значения $\alpha \approx 0.795$, $\beta \approx 0.864 - 2$. (Значение 2 вычитается из-за того, что элементы таблицы включают наряду с промежуточными проходами также начальный и конечный.) К значению, полученному по формуле (9), нужно добавить время перемотки вводной ленты после начального распределения, а именно — $\rho NC\omega_i\tau + \delta$.

Пример 3. Каскадное слияние с прямым чтением. Из табл. 5.4.3–1 следуют значения $\alpha \approx 0.773$, $\beta \approx 0.808 - 2$. Время перемотки сравнительно трудно оценить; возможно, предположение $\pi' \approx \pi$ достаточно точно отражает суть дела. Как и в примере 2, нужно добавить к (9) время начальной перемотки.

Пример 4. Многофазное слияние с расщеплением лент. Из табл. 5.4.2–6 получаем $\alpha \approx 0.752$, $\beta \approx 1.024 - 2$. Происходит совмещение почти всех операций перемотки, за исключением перемотки после начальной установки ($\rho NC\omega_i\tau + \delta$) и двух фаз вблизи конца (36% от $2\rho NC\omega\tau$). Мы можем также вычесть 0.18 из β, так как первая половина фазы совмещается с начальной перемоткой.

Пример 5. Каскадное слияние с совмещением перемоток. Здесь, используя табл. 5.4.3–1 для $T = 5$, получаем $\alpha \approx 0.897$, $\beta \approx 0.800 - 2$. Почти все несовмещенные операции перемотки встречаются непосредственно после начального распределения и после каждого двухпутевого слияния. После точного начального распределения самая длинная лента содержит примерно $1/g$ всех данных, где g есть отношение роста. После каждого двухпутевого слияния объем перемотки в случае шести лент

равен $d_k d_{n-k}$ (см. упр. 5.4.3–5) и можно показать, что в случае T лент объем перемотки после двухпутевых слияний приблизительно равен

$$\big(2/(2T-1)\big)\big(1-\cos\big(4\pi/(2T-1)\big)\big)$$

от всего файла. В нашем случае $T = 5$, что составляет $\frac{2}{9}(1-\cos 80^\circ) \approx 0.184$ от длины файла, и это происходит в $0.946 \ln S + 0.796 - 2$ случаях.

Пример 6. Сбалансированное слияние с обратным чтением. Оно напоминает пример 1, но значительная часть операций перемотки устраняется. Изменение направления обработки ленты от прямого к обратному вызывает некоторые задержки, но они несущественны. С вероятностью 1/2 перед последним проходом понадобится перемотка, поэтому можно взять $\pi' = 1/(2P)$.

Пример 7. Многофазное слияние с обратным чтением. Так как в этом случае выбор с замещением порождает серии, меняющие направление примерно через каждые P раз, то следует заменить (3) другой формулой для S. Достаточно хорошим приближением будет $S = \lceil N(3+1/P)/(6P')\rceil + 1$ (см. упр. 5.4.1–24). Все время перемотки устраняется, и из табл. 5.4.2–1 следует, что $\alpha \approx 0.863$, $\beta \approx 0.921 - 2$.

Пример 8. Каскадное слияние с обратным чтением. Из табл. 5.4.3–1 имеем $\alpha \approx 0.897$, $\beta \approx 0.800 - 2$. Время перемотки по этой таблице можно оценить как удвоенную разность ["проходы с копированием" минус "проходы без копирования"] плюс $1/(2P)$ в том случае, если перед окончательным слиянием необходима перемотка для получения возрастающего порядка.

Пример 9. Осциллирующая сортировка с обратным чтением. В этом случае выбор с замещением должен многократно начинаться и останавливаться; за один раз распределяется от $P-1$ до $2P-1$ серии (в среднем P); средняя длина серий, следовательно, оказывается приблизительно равной $P'(2P-4/3)/P$ и можно оценить $S = \lceil N/\big((2-4/(3P))P'\big)\rceil + 1$. Некоторое время расходуется на переключение от слияния к распределению и обратно; оно приблизительно равно времени, затрачиваемому на чтение P' записей с вводной ленты, а именно — $P'C\omega_i\tau$, и это происходит примерно S/P раз. Время перемотки и время слияния можно оценить, как в примере 6.

Пример 10. Осциллирующая сортировка с прямым чтением. Данный метод нелегко проанализировать, поскольку окончательная фаза "чистки", выполняемая после исчерпания исходных данных, не так эффективна, как предыдущие. Пренебрегая этим трудным аспектом и просто считая, что есть один дополнительный проход, можно оценить время слияния, полагая $\alpha = 1/\ln P$, $\beta = 0$ и $\pi' = \pi/P$. Распределение серий в этом случае несколько иное, так как не используется выбор с замещением; мы устанавливаем $P' = M/C$ и $S = \lceil N/P'\rceil$. Приложив некоторые усилия, можно совместить вычисление, чтение и запись во время распределения, введя дополнительный коэффициент накладных расходов, равный приблизительно $(M+2B)/M$. Время переключения режимов, упомянутое в примере 9, в настоящем случае не нужно учитывать, поскольку переключение совмещается с перемоткой. Итак, оценка времени сортировки имеет вид (9) плюс $2BNC\omega_i\tau/M$.

Из табл. 1 следует, что полученные оценки достаточно хорошо совпадают с реальностью, хотя в нескольких случаях расхождение составляет порядка 50 с. Формулы в примерах 2 и 3 показывают, что каскадному слиянию нужно отдать

Таблица 1

СВОДНАЯ ТАБЛИЦА ОЦЕНОК ВРЕМЕНИ СОРТИРОВКИ

Пример	P	B	P'	S	ω	α	β	α'	β'	(9)	Добавления к (9)	Оценка итога	Реальный итог
1	3	12500	650	79	1.062	0.910	−1.000	0.303	0.000	1064		1064	1076
2	5	8300	734	70	1.094	0.795	−1.136	0.795	−1.136	1010	$\rho NC\omega_i\tau + \delta$	1113	1103
3	5	8300	734	70	1.094	0.773	−1.192	0.773	−1.192	972	$\rho NC\omega_i\tau + \delta$	1075	1127
4	4	10000	700	73	1.078	0.752	−0.994	0.000	0.720	844	$\rho NC\omega_i\tau + \delta$	947	966
5	4	10000	700	73	1.078	0.897	−1.200	0.173	0.129	972		972	992
6	3	12500	650	79	1.062	0.910	−1.000	0.000	0.167	981		981	980
7	4	10000	700	79	1.078	0.863	−1.079	0.000	0.000	922		922	907
8	4	10000	700	73	1.078	0.897	−1.200	0.098	0.117	952		952	949
9	4	10000	700	87	1.078	0.721	−1.000	0.000	0.125	846	$P'SC\omega_i\tau/P$	874	928
10	4	10000	—	100	1.078	0.721	0.000	0.180	0.000	1095	$2BNC\omega_i\tau/M$	1131	1158

предпочтение перед многофазным на шести лентах, тем не менее на практике многофазное слияние лучше! Причина этого кроется в том, что графики, подобные изображенным на рис. 85 (на котором показан случай для пяти лент), ближе к прямым линиям для многофазного алгоритма; каскадный метод превосходит многофазный на шести лентах для $14 \le S \le 15$ и $43 \le S \le 55$ вблизи "точных" каскадных чисел 15 и 55, но многофазное распределение по алгоритму 5.4.2D лучше или, по крайней мере, не хуже для всех остальных $S \le 100$. Каскадный метод предпочтительнее многофазного при $S \to \infty$, но на практике S не может быть слишком большим. Заниженная оценка в примере 9 обусловлена аналогичными обстоятельствами; многофазная сортировка лучше осциллирующей, несмотря на то что, как следует из асимптотического выражения, для больших S осциллирующая сортировка будет наилучшей.

Несколько дополнительных замечаний. Сейчас самое время сделать несколько более или менее случайных замечаний относительно ленточного слияния.

• Приведенные формулы показывают, что сортировка является, в сущности, функцией от произведения N и C, а не N и C по отдельности. За исключением нескольких относительно незначительных соображений (например, что B выбирается кратным C) из наших формул следует, что сортировка миллиона записей по 10 символов займет примерно столько же времени, сколько сортировка 100 000 записей по 100 символов. На самом деле здесь может появиться различие, которое невозможно обнаружить в наших формулах, так как во время выбора с замещением некоторое пространство используется для полей связи. В любом случае размер *ключа* едва ли окажет какое-либо влияние, если только ключи не будут столь длинными и сложными, что внутренние вычисления не смогут угнаться за работой НМЛ.

При длинных записях и коротких ключах соблазнительно выделить ключи, рассортировать их, а затем как-нибудь переставить записи целиком. Но эта идея, кажется, не работает: она может только отсрочить агонию, поскольку процедура окончательной перестановки требует почти столько же времени, сколько потребовала бы общепринятая сортировка методом слияния.

• Разрабатывая программу сортировки, которая должна использоваться многократно, разумно очень тщательно оценить время работы и сравнить теорию с действительными наблюдаемыми характеристиками выполнения. Так как теория

сортировки развита довольно хорошо, эта процедура, как известно, способна внезапно выявить дефекты в оборудовании или программном обеспечении ввода-вывода в существующих системах. Оказывается, обслуживание выполнялось медленнее, чем следовало, и никто этого не замечал, пока данный недостаток не проявился на программе сортировки!

• Наш анализ выбора с замещением был выполнен в предположении, что данные в файле расположены случайно, но файлы, встречающиеся на практике, очень часто уже являются упорядоченными в той или иной степени. (Фактически иногда пользователи обращаются к сортировке файлов, уже упорядоченных ранее, только для того, чтобы убедиться в этом.) Таким образом, опыт даже в бóльшей мере, чем указывают наши формулы, показал, что выбор с замещением предпочтительнее других видов внутренней сортировки. Данное преимущество несколько ослабляется в случае многофазной сортировки с обратным чтением, так как должен быть порожден ряд убывающих серий; на самом деле Р. Л. Гилстад (R. L. Gilstad) (он первым опубликовал описание метода многофазного слияния) первоначально по этой причине отверг метод обратного чтения. Но позднее он заметил, что чередование направлений будет все же давать длинные возрастающие серии. Кроме того, многофазный метод с обратным чтением — это единственный стандартный метод, который благосклонен к убывающим входным файлам в той же степени, что и к возрастающим.

• Другое преимущество выбора с замещением состоит в том, что этот метод допускает совмещение процессов чтения, записи и вычислений. Если бы мы просто выполняли внутреннюю сортировку очевидным способом — заполнили память, расссортировали данные в ней и затем записывали данные по мере того, как память заполняется новым содержимым, — то проход распределения занял бы примерно вдвое больше времени.

Из рассмотренных нами методов внутренней сортировки еще только один можно приспособить к одновременному чтению, записи и вычислениям — пирамидальную сортировку. (Эта идея была использована при подготовке примера 10 диаграммы А.) Предположим для удобства, что внутренняя память содержит 1 000 записей, а каждый блок на ленте — по 100. Действовать можно следующим образом (через $B_2 \ldots B_{10}$ обозначено содержимое памяти, разделенной на 10 блоков по 100 записей).

Шаг 0. Заполнить память и сделать так, чтобы элементы $B_2 \ldots B_{10}$ удовлетворяли неравенствам пирамиды (с наименьшим элементом в вершине).

Шаг 1. Свести $B_1 \ldots B_{10}$ в пирамиду, затем выбрать наименьшие 100 записей и переписать их в B_{10}.

Шаг 2. Записать из B_{10}; в то же время выбрать наименьшие 100 записей из $B_1 \ldots B_9$ и поместить их в B_9.

Шаг 3. Прочитать в B_{10} и записать в B_9; в то же время выбрать наименьшие 100 записей из $B_1 \ldots B_8$ и поместить их в B_8.

$\vdots$

Шаг 9. Прочитать в B_4 и записать из B_3; в то же время выбрать наименьшие 100 записей из $B_1\, B_2$, поместить их в B_2 и сделать так, чтобы неравенства пирамиды были справедливы для $B_5 \ldots B_{10}$.

Шаг 10. Прочитать в B_3 и записать из B_2, сортируя B_1; в то же время сделать так, чтобы неравенства пирамиды были справедливы для $B_4 \ldots B_{10}$.

Шаг 11. Прочитать в B_2 и записать из B_1; в то же время сделать так, чтобы $B_3 \ldots B_{10}$ удовлетворяли неравенствам пирамиды.

Шаг 12. Прочитать в B_1, делая так, чтобы $B_2 \ldots B_{10}$ удовлетворяли неравенствам пирамиды. Вернуться к шагу 1. ▌

• Мы предполагаем, что число сортируемых записей N заранее неизвестно. На самом же деле большинство компьютеров позволяет постоянно следить за числом записей во всех файлах и можно считать, что наша вычислительная система способна сообщить значение N. Насколько существенно нам бы это помогло? К сожалению, не очень! Мы видели, что выбор с замещением весьма выгоден, но он ведет к непредсказуемому числу начальных серий. В сбалансированном слиянии мы могли бы использовать информацию о N для установления такого размера буфера B, чтобы S оказалось, скорее всего, чуть меньше степени P; в многофазном распределении с оптимальным размещением фиктивных серий мы могли бы использовать информацию о N чтобы решить, какой уровень выбрать (см. табл. 5.4.2–2).

• НМЛ часто оказываются наименее надежными устройствами в вычислительной технике. Следовательно, можно принять за аксиому, что *исходная вводная лента ни в коем случае не должна изменяться, пока не станет известно, что вся сортировка удовлетворительно завершена*. В некоторых примерах на диаграмме А существует досадное "время, пока оператор сменит ленту", но было бы слишком рискованно затирать исходные данные ввиду возможности появления какого-либо сбоя во время длинной сортировки.

• При переходе от прямой записи к обратному чтению мы можем сэкономить некоторое время, вовсе не записывая последний буфер на ленту; он в любом случае будет вновь прочитан. Диаграмма А показывает, что этот прием в действительности экономит сравнительно немного времени, за исключением случая осциллирующей сортировки, когда направления меняются часто.

• Хотя в большинстве вычислительных систем имеется достаточно много НМЛ, было бы слишком рискованно все их использовать в процессе сортировки. Рекомендуемое процентное отношение находится в диапазоне между $\log_P S$ и $\log_{P+1} S$ и не очень велико при достаточно большом P. Более высокий порядок сортировки влечет за собой, как правило, использование небольших по размеру блоков. Подумайте также о бедном операторе компьютера, который должен устанавливать все эти рабочие ленты! С другой стороны, в упр. 12 описан интересный способ использования дополнительных НМЛ, группируемых так, чтобы совместить время ввода и вывода без увеличения порядка слияния.

• При использовании компьютеров, подобных MIX, которые имеют фиксированный и довольно маленький размер блоков, для слияния едва ли требуется много внутренней памяти. Здесь осциллирующая сортировка более предпочтительна, потому что становится возможным сохранять дерево выбора с замещением в памяти во время слияния. На самом деле в этом случае можно усовершенствовать осциллирующую сортировку (как предложил К. Дж. Белл (C. J. Bell) в 1962 году), сливая новую начальную серию с выводом каждый раз. когда выполняется слияние с рабочих лент!

• Мы видели, что файлы на нескольких бобинах должны сортироваться последовательно бобина за бобиной, чтобы избежать чрезмерной работы, связанной с перестановкой лент. Иногда такого рода приложения называются многобобинными. Фактически сбалансированное слияние с шестью лентами, если оно тщательно запрограммировано, может сортировать *три* бобины до момента окончательного слияния.

Для слияния относительно большого числа отдельно рассортированных бобин лучше всего использовать дерево слияния с минимальной длиной пути (ср. с разделом 5.4.4). Этот подход впервые был реализован Э. Г. Френдом (E. H. Friend) [*JACM* **3** (1956), 166–167] и затем — У. Г. Буржем (W. H. Burge) [*Information and Control* **1** (1958), 181–197], которые отметили, что оптимальный способ слияния серий данных (возможно, неравных длин) получается путем построения дерева с минимальной *взвешенной* длиной пути и использования длины серий в качестве весов (см. разделы 2.3.4.5 и 5.4.9), если пренебречь временем установки лент. Но файлы, занимающие несколько бобин, вероятно, следует хранить на дисках или другом запоминающем устройстве большой емкости, а не на лентах.

• В нашем обсуждении мы, не особенно задумываясь, предполагали, что имеется возможность использовать непосредственно инструкции ввода-вывода и что никакой сложный системный интерфейс не мешает нам использовать ленты с такой эффективностью, на какую рассчитывали конструкторы оборудования. Эти идеальные предположения позволили нам проникнуть в суть проблем слияния, и они могут дать некоторый подход к разработке соответствующих операционных систем. Однако следует понимать, что мультипрограммирование и мультипроцессирование могут значительно усложнить ситуацию.

• Обсуждаемые нами вопросы впервые были рассмотрены в работах E. H. Friend, *JACM* **3** (1956), 134–168; W. Zoberbier, *Elektronische Datenverarbeitung* **5** (1960), 28–44, и M. A. Goetz, *Digital Computer User's Handbook* (New York: McGraw-Hill, 1967), 1.292–1.320.

Резюме. Мы можем следующим образом вкратце подытожить все, что узнали о сравнении различных схем слияния.

Теорема А. *Трудно решить, какая схема слияния является наилучшей в конкретной ситуации.* ▌

Примеры, которые мы видели на диаграмме А, показывают, как 100 000 записей по 100 символов (или миллион записей по 10 символов), расположенных в случайном порядке, могли бы быть рассортированы с помощью шести лент при достаточно реалистических предположениях. Эти данные занимают около половины ленты и могут быть рассортированы приблизительно за 15–19 мин при использовании НМЛ `MIXT`. Однако существующее ленточное оборудование сильно различается по возможностям, и время выполнения такой работы на разных машинах изменяется в диапазоне приблизительно от 4 мин до 2 ч. В наших примерах около 3 мин расходуется на начальное распределение серий и внутреннюю сортировку, около $4\frac{1}{2}$ мин — на окончательное слияние и перемотку выводной ленты и приблизительно $7\frac{1}{2}$–$11\frac{1}{2}$ мин — на промежуточные стадии слияния.

Для шести лент, которые нельзя читать в обратном направлении, наилучшим методом сортировки при наших предположениях было многофазное слияние с рас-

щеплением лент (пример 4), а для НМЛ, допускающих обратное чтение, наилучшим оказался многофазный метод с обратным чтением со сложным размещением фиктивных серий (пример 7). Осциллирующая сортировка (пример 9) занимает в нашей табели о рангах второе место. В обоих случаях каскадное слияние является более простым и лишь незначительно медленнее (примеры 5 и 8). В случае прямого чтения обычное сбалансированное слияние (пример 1) оказалось удивительно эффективным, частично из-за удачного сочетания параметров в данном конкретном примере, а частично из-за того, что при этом тратится сравнительно мало времени на перемотку.

Положение несколько изменилось бы, если бы в нашем распоряжении было другое число НМЛ.

Генераторы сортировки. В условиях большого разнообразия характеристик данных и оборудования почти невозможно написать единственную программу внешней сортировки, которая была бы удовлетворительной в подавляющем большинстве случаев. Также весьма трудно создать программу, которая в реальных условиях эффективно работает с лентами. Следовательно, разработка программного обеспечения сортировки — самостоятельная задача, требующая большой работы. *Генератор сортировки* — это программа, которая, основываясь на параметрах, описывающих формат данных и конфигурацию оборудования, порождает машинную программу, специально приспособленную к конкретному применению сортировки. Подобная программа часто связана с такими языками высокого уровня, как COBOL и PL/I, или она может быть написана как набор макроопределений для использования совместно с макроассемблером.

Одной из особенностей, обычно обеспечиваемых генератором сортировки, является возможность вставлять собственные команды — особые инструкции, которые должны включатся в первый и последний проходы программы сортировки. Собственные команды первого прохода обычно используются для некоторой коррекции исходных записей, часто сокращая их или незначительно удлиняя, чтобы привести к форме, более простой для сортировки. Пусть, например, исходные записи должны быть рассортированы по девятисимвольному ключу, изображающему дату в формате "месяц-день-год":

```
JUL041776  OCT311517  NOV051605  JUL141789  NOV071917.
```

Трехбуквенные коды месяцев можно найти в таблице и заменить числами, причем наиболее значащие поля могут быть помещены слева:

```
17760704  15171031  16051105  17890714  19171107.
```

Это уменьшает длину записей и упрощает последующие сравнения. (Код ключей мог бы быть сделан даже более компактным.) Собственные команды последнего прохода могут использоваться для восстановления исходного формата и/или для внесения других желательных изменений в файл, и/или для вычисления какой-либо функции от выводных записей. Алгоритмы слияния, которые мы изучили, организованы таким образом, что последний проход легко отличить от остальных фаз слияния. Заметим, что если имеются собственные команды, то должно быть, по крайней мере, два прохода по файлу, даже если он первоначально находился в нуж-

ном порядке. Собственные команды, изменяющие размер записей, могут затруднить совмещение некоторых операций ввода-вывода в осциллирующей сортировке.

Генераторы сортировки заботятся о таких системных деталях, как соглашения о метках на лентах; они также часто обеспечивают подсчет контрольной суммы или иные проверки того, что никакая часть файла не пропала и не изменилась. Иногда имеются средства для остановки сортировки в удобных местах и возобновления ее позднее. Самые высококачественные генераторы позволяют записям иметь динамически меняющиеся длины [см. D. J. Waks, *CACM* **6** (1963), 267–272].

***Слияние с меньшим числом буферов.** Мы видели, что $2P + 2$ буферов достаточно для поддержания быстрого движения лент в течение P-путевого слияния. В завершение этого раздела проведем математический анализ времени слияния в том случае, когда в нашем распоряжении имеется *меньше* $2P + 2$ буферов.

Очевидно, что желательно иметь два буфера вывода — можно будет выполнять запись из одного буфера, образуя в это же время следующий блок вывода в другом. Поэтому можно вообще не рассматривать вопрос вывода и заняться только вводом.

Допустим, имеется $P + Q$ буферов ввода, где $1 \le Q \le P$. Воспользуемся для описания нашей ситуации моделью, предложенной в работе L. J. Woodrum, *IBM Systems J.* **9** (1970), 118–144. Чтение одного блока ленты занимает одну единицу времени. Обозначим через p_0 вероятность того, что в течение этого времени ни один из буферов ввода не станет пустым, через p_1 — что один буфер станет пустым, через $p_{\ge 2}$ — что два или больше буферов станут пустыми и т. д. По завершении чтения ленты мы оказываемся в одном из $Q + 1$ состояний.

Состояние 0. Q буферов пусты. Мы начинаем читать блок подходящего файла в один из них, используя метод прогнозирования, описанный ранее в этом разделе. Через один такт времени мы переходим в состояние 1 с вероятностью p_0, в противном случае остаемся в состоянии 0.

Состояние 1. $Q - 1$ буферов пусты. Начинаем выполнять чтение в один из них, предсказывая подходящий файл. Через один такт времени переходим в состояние 2 с вероятностью p_0, в состояние 1 с вероятностью p_1 и в состояние 0 с вероятностью $p_{\ge 2}$.

$\vdots$

Состояние $Q - 1$. Один буфер пуст. Начинаем выполнять чтение в него, предсказывая подходящий файл. Через один такт времени переходим в состояние Q с вероятностью p_0, в состояние $Q - 1$ с вероятностью p_1, ..., в состояние 1 с вероятностью p_{Q-1} и в состояние 0 с вероятностью $p_{\ge Q}$.

Состояние Q. Все буфера заполнены. Чтение лент останавливается в среднем на μ тактов времени, и затем мы переходим в состояние $Q - 1$.

Начинаем с состояния 0. Модель этой ситуации соответствует *марковскому процессу* (см. упр. 2.3.4.2–26), который можно проанализировать с помощью производящих функций следующим интересным способом. Пусть z — произвольный параметр, и предположим, что каждый раз, когда мы решаем осуществлять чтение с ленты, делаем это с вероятностью z, а с вероятностью $1 - z$ завершаем выполнение алгоритма. Пусть $g_Q(z) = \sum_{n\ge 0} a_n^{(Q)} z^n (1 - z)$ — среднее число появлений в этом процессе состояния Q; отсюда следует, что $a_n^{(Q)}$ — это среднее число появлений

состояния Q, если было прочитано ровно n блоков. Тогда $n + a_n^{(Q)}\mu$ — среднее суммарное время, затраченное на ввод и вычисления. Если бы имелось полное совмещение, как в алгоритме с $(2P+2)$ буферами, то суммарное время включало бы только n единиц, так что $a_n^{(Q)}\mu$ представляет время задержки чтения.

Пусть A_{ij} — вероятность того, что мы переходим из состояния i в состояние j в этом процессе при $0 \le i, j \le Q+1$, где $Q+1$ — новое состояние "остановки". Например, при малых Q матрицы A будут следующими:

$$Q = 1: \qquad \begin{pmatrix} p_{\ge 1}z & p_0 z & 1-z \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix},$$

$$Q = 2: \qquad \begin{pmatrix} p_{\ge 1}z & p_0 z & 0 & 1-z \\ p_{\ge 2}z & p_1 z & p_0 z & 1-z \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix},$$

$$Q = 3: \qquad \begin{pmatrix} p_{\ge 1}z & p_0 z & 0 & 0 & 1-z \\ p_{\ge 2}z & p_1 z & p_0 z & 0 & 1-z \\ p_{\ge 3}z & p_2 z & p_1 z & p_0 z & 1-z \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}.$$

Из упр. 2.3.4.2–26(b) имеем, что $g_Q(z) = \mathrm{cofactor}_{Q0}(I-A)/\det(I-A)$. Так, например, если $Q = 1$, имеем

$$g_1(z) = \det\begin{pmatrix} 0 & -p_0 z & z-1 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \Bigg/ \det\begin{pmatrix} 1-p_{\ge 1}z & -p_0 z & z-1 \\ -1 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$= \frac{p_0 z}{1 - p_1 z - p_0 z} = \frac{p_0 z}{1-z} = \sum_{n \ge 0} n p_0 z^n (1-z),$$

так что $a_n^{(1)} = np_0$. Это, конечно, *очевидно заранее*, так как при $Q = 1$ задача очень проста. Аналогичное вычисление для $Q = 2$ (см. упр. 14) дает менее очевидную формулу:

$$a_n^{(2)} = \frac{p_0^2 n}{1-p_1} - \frac{p_0^2(1-p_1^n)}{(1-p_1)^2}. \tag{10}$$

В общем случае можно показать, что $a_n^{(Q)}$ имеет вид $\alpha^{(Q)}n + O(1)$ при $n \to \infty$, где константу $\alpha^{(Q)}$ не слишком трудно вычислить (см. упр. 15). Как оказывается, $\alpha^{(3)} = p_0^3 / \bigl((1-p_1)^2 - p_0 p_2\bigr)$.

Исходя из природы слияния, довольно разумно предположить, что $\mu = 1/P$, и мы имеем биномиальное распределение

$$p_k = \binom{P}{k}\left(\frac{1}{P}\right)^k \left(\frac{P-1}{P}\right)^{P-k}.$$

Например, если $P = 5$, то $p_0 = .32768$, $p_1 = .4096$, $p_2 = .2048$, $p_3 = .0512$, $p_4 = .0064$ и $p_5 = .00032$; следовательно, $\alpha^{(1)} \approx 0.328$, $\alpha^{(2)} \approx 0.182$ и $\alpha^{(3)} \approx 0.125$. Другими словами, если мы используем $5+3$ вводных буферов вместо $5+5$, то можно ожидать дополнительного времени "задержки чтения" порядка $0.125/5 \approx 2.5\%$.

Конечно, эта модель — только очень грубое приближение. Мы знаем, что при $Q = P$ вообще нет времени задержки, но если судить по модели, то есть. Дополнительное время задержки чтения для меньших Q почти точно уравновешивает выигрыш в накладных расходах, получаемый от использования более крупных блоков, так что выбор простой схемы с $Q = P$ кажется оправданным.

УПРАЖНЕНИЯ

1. [*13*] Выведите формулу для вычисления точного числа символов на ленте, если в каждом блоке содержится n символов. Считайте, что лента могла бы вместить ровно 23 000 000 символов, если бы не было междублочных промежутков.

2. [*15*] Объясните, почему первый буфер файла 2 в строке 6 на рис. 84 совсем пуст.

3. [*20*] Будет ли алгоритм F работать должным образом, если вместо $2P$ буферов ввода имеется только $2P-1$? Если да, докажите это; если нет, приведите пример, когда алгоритм не выполняется.

4. [*20*] Как изменить алгоритм F, чтобы он работал и при $P = 1$?

▶ **5.** [*21*] Если в различных файлах имеются равные ключи, необходимо в процессе прогнозирования действовать очень аккуратно. Объясните, почему. Покажите, как избежать трудностей, если более строго определить операции слияния и прогнозирования в алгоритме F.

6. [*22*] Какие изменения при работе с $T+1$ лентами следует сделать в алгоритме 5.4.3C, чтобы преобразовать его в алгоритм каскадного слияния с *совмещением перемотки*?

▶ **7.** [*26*] Начальное распределение в примере 7 диаграммы A порождает

$$(A_1D_1)^{11} \qquad D_1(A_1D_1)^{10} \qquad D_1(A_1D_1)^{9} \qquad D_1(A_1D_1)^{7}$$

на лентах 1–4, где $(A_1D_1)^7$ означает $A_1D_1A_1D_1A_1D_1A_1D_1A_1D_1A_1D_1A_1D_1$. Покажите, как вставить дополнительные серии A_0 и D_0 наилучшим из возможных способов (в том смысле, что общее число обрабатываемых во время слияния начальных серий будет минимальным), чтобы привести распределение к следующему виду:

$$A(DA)^{14} \qquad (DA)^{28} \qquad (DA)^{26} \qquad (DA)^{22}.$$

Указание. Чтобы сохранить четность, необходимо вставлять серии A_0 и D_0 в виде соседних пар. Числа слияния для каждой начальной серии могут быть подсчитаны, как в упр. 5.4.4–5; здесь появляется некоторое упрощение, так как соседние серии всегда имеют соседние числа слияния.

8. [*20*] На диаграмме A видно, что большинство схем начального распределения серий (за исключением начального распределения для каскадного слияния) имеет тенденцию к помещению последовательных серий на различные ленты. Если бы последовательные серии попали на одну ленту, то мы могли бы сэкономить стартстопное время. Можно ли поэтому считать продуктивной идею изменения алгоритмов распределения так, чтобы они реже переключали ленты?

▶ **9.** [*22*] Оцените, сколько времени заняло бы выполнение многофазного алгоритма с обратным чтением из показанных на диаграмме A, если бы для сортировки использовались все $T = 6$ лент, а не $T = 5$, как в примере 7. Было ли разумно избегать использования вводной ленты?

10. [*M23*] Используя анализ, проведенный в разделах 5.4.2 и 5.4.3, покажите, что длина каждой перемотки во время стандартного многофазного слияния с шестью лентами или каскадного слияния редко превышает 54% длины файла (исключая начальную и конечную перемотки, которые охватывают весь файл).

11. [*23*] Откорректировав соответствующие элементы табл. 1, оцените, сколько времени заняло бы выполнение первых девяти примеров на диаграмме А, если бы мы имели двухскоростную перемотку (быструю и медленную). Считайте, что $\rho = 1$, если лента заполнена меньше чем на одну четверть, а для более заполненной ленты время перемотки равно приблизительно 5 с плюс то время, которое получилось бы при $\rho = \frac{1}{5}$. Измените пример 8 так, чтобы в нем использовалось каскадное слияние *с копированием*, поскольку перемотка и прямое чтение в этом случае происходят медленнее копирования. [*Указание.* Используйте результат упр. 10.]

12. [*40*] Рассмотрим разбиение шести лент на три пары лент, где каждая пара играет роль одной ленты в многофазном слиянии с $T = 3$. Одна лента каждой пары будет содержать блоки $1, 3, 5, \ldots$, а другая — блоки $2, 4, 6, \ldots$; таким способом мы, по существу, добиваемся того, чтобы во все время слияния две вводные и две выводные ленты оставались активными, причем эффективная скорость слияния удваивается.

a) Найдите подходящий способ распространения алгоритма F на этот случай.

b) Оцените общее время выполнения, которое получилось бы, если бы этот метод использовался для сортировки 100,000 записей по 100 символов, рассмотрев случаи как прямого, так и обратного чтения.

13. [*20*] Может ли метод осциллирующей сортировки с пятью лентами в том виде, в котором он определен в алгоритме 5.4.5B, использоваться для сортировки четырех полных бобин исходных данных до момента окончательного слияния?

14. [*M19*] Выведите (10).

15. [*HM29*] Докажите, что $g_Q(z) = h_Q(z)/(1-z)$, где $h_Q(z)$ является рациональной функцией z, не имеющей особых точек внутри единичного круга; следовательно, $a_n^{(Q)} = h_Q(1)n + O(1)$ при $n \to \infty$. В частности, покажите, что

$$h_3(1) = \det\begin{pmatrix} 0 & -p_0 & 0 & 0 \\ 0 & 1-p_1 & -p_0 & 0 \\ 0 & -p_2 & 1-p_1 & -p_0 \\ 1 & 0 & 0 & 0 \end{pmatrix} \Bigg/ \det\begin{pmatrix} 1 & -p_0 & 0 & 0 \\ 1 & 1-p_1 & -p_0 & 0 \\ 1 & -p_2 & 1-p_1 & -p_0 \\ 0 & 0 & -1 & 1 \end{pmatrix}.$$

16. [*41*] Детально изучите сортировку 100,000 записей по 100 символов, нарисуйте диаграммы, подобные диаграмме А, в предположении, что имеется 3, 4 и 5 лент.

*5.4.7. Внешняя поразрядная сортировка

В предыдущих разделах рассматривалась сортировка методом слияния файлов на лентах. Но существует и другой способ сортировки на лентах, основанный на принципе поразрядной сортировки, который ранее использовался в механических сортировальных машинах для перфокарт (см. раздел 5.2.5). Этот метод иногда называют распределяющей сортировкой, сортировкой по колонкам, карманной сортировкой, цифровой сортировкой, сортировкой методом разделения и т. д. Он, как оказывается, по существу, *противоположен* слиянию!

Предположим, например, что в нашем распоряжении имеются четыре ленты, а ключей может быть только восемь: 0, 1, 2, 3, 4, 5, 6, 7. Если исходные данные

содержатся на ленте T1, то начнем с переписи всех четных ключей на T3 и всех нечетных — на T4.

	T1	T2	T3	T4
Дано	$\{0,1,2,3,4,5,6,7\}$	—	—	—
Проход 1	—	—	$\{0,2,4,6\}$	$\{1,3,5,7\}$

Теперь перематываем ленты и читаем сначала T3, а затем — T4, переписывая {0, 1, 4, 5} на T1 и {2, 3, 6, 7} на T2.

Проход 2	$\{0,4\}\{1,5\}$	$\{2,6\}\{3,7\}$	—	—

(Строка "$\{0,4\}\{1,5\}$" обозначает файл, содержащий записи только с ключами 0 и 4, за которыми следуют записи только с ключами 1 и 5. Заметим, что на T1 теперь находятся те ключи, средний двоичный разряд которых содержит 0.) После еще одной перемотки и распределения ключей 0, 1, 2, 3 на T3 и ключей 4, 5, 6, 7 на T4 получаем следующее.

Проход 3	$\{0\}\{1\}\{2\}\{3\}$	$\{4\}\{5\}\{6\}\{7\}$

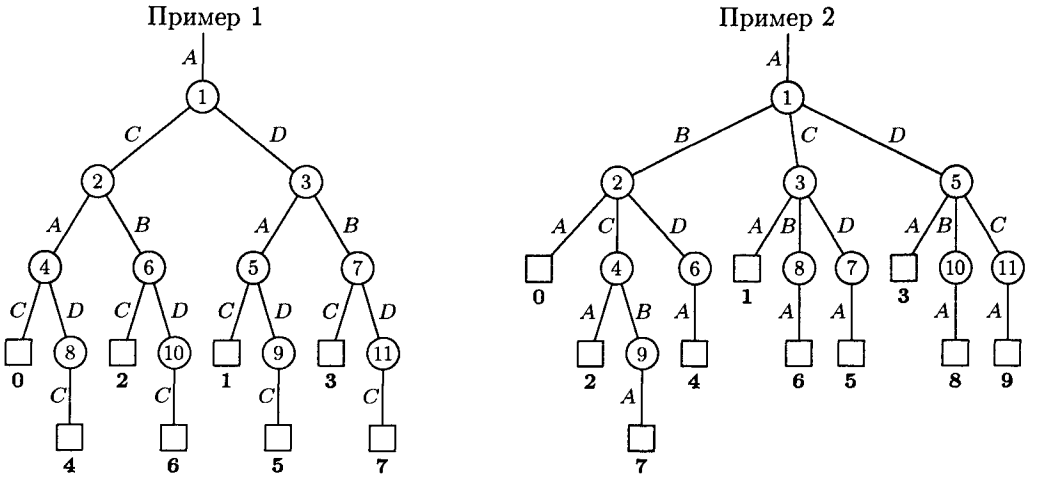
После копирования T4 в конец T3 работа завершается. В общем случае для ключей в диапазоне от 0 до $2^k - 1$ можно аналогичным образом рассортировать файл за k проходов, после которых следует фаза окончательной "сборки". На этом этапе примерно половина данных копируется с одной ленты на другую. Имея шесть лент, можно так же использовать представления по основанию 3 для сортировки ключей от 0 до $3^k - 1$ за k проходов.

Существуют модификации этого метода с частичными проходами. Предположим, например, что допускается десять ключей $\{0,1,\ldots,9\}$, и рассмотрим следующую процедуру, авторство которой принадлежит Р. Л. Эшенхерсту (R. L. Ashenhurst) [*Theory of Switching*, Progress Report BL-7 (Harvard Univ. Comp. Laboratory: May, 1954), I.1–I.76].

Фаза	T1	T2	T3	T4	Проход
	$\{0,1,\ldots,9\}$	—	—	—	
1	—	$\{0,2,4,7\}$	$\{1,5,6\}$	$\{3,8,9\}$	1.0
2	$\{0\}$	—	$\{1,5,6\}\{2,7\}$	$\{3,8,9\}\{4\}$	0.4
3	$\{0\}\{1\}\{2\}$	$\{6\}\{7\}$	—	$\{3,8,9\}\{4\}\{5\}$	0.5
4	$\{0\}\{1\}\{2\}\{3\}$	$\{6\}\{7\}\{8\}$	$\{9\}$	$\{4\}\{5\}$	0.3
C	$\{0\}\{1\}\{2\}\{3\}\{4\}\ldots\{9\}$				0.6
					2.8

Здесь C представляет фазу сборки. Если каждое значение ключа встречается примерно в одном из десяти случаев, то эта процедура для сортировки десяти ключей затрачивает только 2.8 прохода, в то время как в первом примере требуется 3.5 прохода для сортировки всего восьми ключей. Таким образом, продуманая схема распределения влечет за собой значительное различие в показателях эффективности для поразрядной сортировки точно так же, как для слияния.

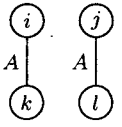
Схемы распределения из предыдущих примеров представим, как и обычно, древовидными структурами.



Круглые внутренние узлы этих деревьев пронумерованы 1, 2, 3, ... в соответствии с шагами процесса 1, 2, 3, ... Имена лент A, B, C, D (вместо T1, T2, T3, T4) помещены рядом с ребрами дерева, чтобы указать, куда попадают записи. Квадратные внешние узлы изображают фрагменты файла, содержащие только один ключ, и этот ключ выделен полужирным шрифтом под соответствующим узлом. Ребра, расположенные над квадратными узлами, помечены именем выводной ленты (C в первом примере, A — во втором).

Таким образом, на шаге 3 примера 1 выполняется считывание с ленты D и запись ключей 1 и 5 на ленту A и ключей 3 и 7 на ленту B. Нетрудно видеть, что число выполняемых проходов равно *длине внешнего пути* дерева, деленной на число внешних узлов в предположении, что все ключи равновероятны.

В связи с последовательной природой ленты и в соответствии с правилом "первым включается — первым исключается", которому подчиняется прямое чтение, нельзя взять за основу схемы распределения *любое* помеченное дерево. В дереве примера 1 данные записываются на ленту A на шагах 2 и 3; данные, записанные в течение шага 2, необходимо использовать раньше данных, записанных в течение шага 3. В общем случае, если осуществляется запись на ленту в течение шагов i и j, где $i < j$, первыми следует использовать данные, записанные в течение шага i. Если дерево содержит две ветви вида



$$, \qquad i < j,$$

то должно выполняться условие $k < l$. Кроме того, нельзя ничего записывать на ленту A *между* шагами k и l, поскольку между чтением и записью необходима перемотка.

Те читатели, которые внимательно проработали упражнения из раздела 5.4.4, сразу поймут, что допустимые деревья для поразрядной сортировки с прямым чтением на T лентах — это в точности *сильные T-fifo-деревья*, описывающие сортировку методом *слияния* на T лентах с прямым чтением! (См. упр. 5.4.4–20.) Единственное различие заключается в том, что все внешние узлы рассматриваемых здесь

деревьев помечены одной и той же лентой. Мы могли бы снять это ограничение, предположив, что существует окончательная фаза "сборки", на которой все записи переносятся на выводную ленту, или могли бы добавить его к правилам для T-fifo-деревьев, потребовав, чтобы начальный распределительный проход сортировки методом слияния был явно выражен в соответствующем дереве слияния.

Иными словами, *каждой схеме слияния соответствует схема распределения* и *каждой схеме распределения соответствует схема слияния.* По некотором размышлении это становится понятным. Рассмотрим сортировку методом слияния, делающую все наоборот, т. е. разделяющую окончательный выводной файл на подфайлы, которые, в свою очередь, разделяются на другие подфайлы, и т. д. Наконец, разделим файл на S серий. Подобная схема может применяться к лентам тогда и только тогда, когда допустима соответствующая схема распределения для поразрядной сортировки S ключей. Эта двойственность слияния и распределения почти точна; она не выполняется только в одном отношении: данные с вводной ленты должны сохраняться в разное время.

Пример с восемью ключами, рассмотренный в начале этого раздела, очевидно, является двойственным сбалансированному слиянию на четырех лентах. Пример с десятью ключами и частичными проходами соответствует следующей схеме слияния десяти серий (если скрыть фазы копирования — шаги 6–11 в дереве).

	T1	T2	T3	T4
Начальное распределение	1^4	1^3	1^1	1^2
Шаг дерева 5	1^3	1^2	—	$1^2 3^1$
Шаг дерева 4	1^2	1^1	2^1	$1^2 3^1$
Шаг дерева 3	1^1	—	$2^1 3^1$	$1^1 3^1$
Шаг дерева 2	—	4^1	3^1	3^1
Шаг дерева 1	10^1	—	—	—

Если сравнить ее с поразрядной сортировкой, то очевидно, что оба метода имеют, в сущности, одну и ту же структуру, но обратны во времени и имеют обратное расположение содержимого на лентах: $1^2 3^1$ (две серии длиной 1 каждая, за которыми следует одна серия длиной 3) соответствует $\{3, 8, 9\}\{4\}\{5\}$ (два подфайла, содержащие по 1 ключу, перед которыми расположен один подфайл, содержащий 3 ключа).

Двигаясь в другую сторону, можно, в принципе, построить поразрядную сортировку, двойственную многофазному слиянию, каскадному слиянию и т. д. Например, многофазному слиянию 21 серии на трех лентах, изображенному в начале раздела 5.4.2, соответствует следующая интересная поразрядная сортировка.

Фаза	T1	T2	T3
	{0,1,...,20}	—	—
1	—	{0,2,4,5,7,9,10,12,13,15,17,18,20}	{1,3,6,8,11,14,16,19}
2	{0,5,10,13,18}	—	{1,3,6,8,11,14,16,19} {2,4,7,9,12,15,17,20}
3	{0,5,10,13,18}{1,6,11,14,19} {2,7,12,15,20}	{3,8,16}{4,9,17}	—
4	—	{3,8,16}{4,9,17}{5,10,18} {6,11,19}{7,12,20}	{0,13}{1,14}{2,15}
5	{8}{9}{10}{11}{12}	—	{0,13}{1,14}{2,15} {3,16}...{7,20}
6	{8}{9}{10}{11}{12}{13}...{20}	{0}{1}...{7}	—

Правило распределения, согласно которому ключи располагаются на лентах на каждом шаге, кажется магическим, но на самом деле оно имеет простую связь с системой счисления, использующей числа Фибоначчи! (См. упр. 2.)

Обратное чтение. Двойственность поразрядной сортировки и слияния применима также к алгоритмам, читающим ленту в обратном направлении. Мы определили T-lifo-деревья в разделе 5.4.4, и нетрудно видеть, что они подходят для поразрядной сортировки в той же мере, что и для сортировки методом слияния.

Поразрядная сортировка с обратным чтением была фактически рассмотрена Джоном Мочли (John Mauchly) еще в 1946 году в одной из первых опубликованных работ по сортировке вообще (см. раздел 5.5). Мочли на самом деле предложил следующую схему сортировки.

Фаза	T1	T2	T3	T4
	—	$\{0,1,2,\ldots,9\}$	—	—
1	$\{4,5\}$	—	$\{2,3,6,7\}$	$\{0,1,8,9\}$
2	$\{4,5\}\{2,7\}$	$\{3,6\}$	—	$\{0,1,8,9\}$
3	$\{4,5\}\{2,7\}\{0,9\}$	$\{3,6\}\{1,8\}$	—	—
4	$\{4,5\}\{2,7\}$	$\{3,6\}\{1,8\}$	$\{9\}$	$\{0\}$
. . .	. . .	. . .	. . .	. . .
8	—	—	$\{9\}\{8\}\{7\}\{6\}\{5\}$	$\{0\}\{1\}\{2\}\{3\}\{4\}$
C	—	—	—	$\{0\}\{1\}\{2\}\{3\}\{4\}\{5\}\ldots\{9\}$

Эта схема не является самой эффективной среди всех возможных, но она интересна тем, что показывает: методы с частичными проходами рассматривались применительно к поразрядной сортировке еще в 1946 году, хотя в литературе по слиянию они появились лишь около 1960 года.

Эффективная конструкция схем распределения с обратным чтением была предложена в работе A. Bayes, *CACM* **11** (1968), 491–493. Пусть дано $P+1$ лент и S ключей; разделите ключи на P подфайлов, каждый из которых содержит $\lfloor S/P \rfloor$ или $\lceil S/P \rceil$ ключей, и примените эту процедуру рекурсивно к каждому подфайлу. Если $S < 2P$, то один подфайл должен состоять из единственного наименьшего ключа; его и следует записать на выводную ленту. (Общая конструкция с прямым порядком Р. М. Карпа, описанная в конце раздела 5.4.4, включает этот метод как частный случай.)

Обратное чтение несколько усложняет слияние, поскольку оно обращает порядок серий. Соответствующий эффект возникает и при поразрядной сортировке. Результат оказывается устойчивым или неустойчивым в зависимости от того, какой уровень достигнут в дереве. После поразрядной сортировки с обратным чтением, когда одни внешние узлы находятся на четных уровнях, а другие — на нечетных, для одних ключей относительный порядок различных записей с одинаковыми ключами будет *совпадать* с первоначальным порядком, но для других он будет *противоположен* исходному (см. упр. 6).

Осциллирующая сортировка методом слияния также имеет свою пару в этой двойственности. При *осциллирующей поразрядной сортировке* мы продолжаем разделять ключи, пока не достигнем подфайлов, содержащих только один ключ или достаточно малых, чтобы поддаваться внутренней сортировке. Такие подфайлы сортируются и записываются на выводную ленту, затем процесс разделения возоб-

новляется. Например, если имеются три рабочие ленты и одна выводная и если ключи являются двоичными числами, мы можем сначала поместить ключи вида $0x$ на ленту T1, а ключи $1x$ — на ленту T2. Если на ленте T1 больше записей, чем может поместиться в памяти, то вновь просматриваем ее и помещаем $00x$ на T2 и $01x$ на T3. Теперь, если подфайл $00x$ достаточно короткий, выполняем его внутреннюю сортировку, выводим результат, а затем начинаем обработку подфайла $01x$. Подобный метод был назван Э. Х. Френдом каскадной псевдопоразрядной сортировкой [*JACM* **3** (1956), 157–159]; более подробно его разработали Х. Нэглер (H. Nagler) [*JACM* **6** (1959), 459–468], который дал ему красочное название — "метод двуглавого змия", и Ч. Х. Годетт (C. H. Gaudette) [*IBM Tech. Disclosure Bull.* **12** (April, 1970), 1849–1853].

Имеет ли поразрядная сортировка преимущество перед слиянием. Одним важным следствием принципа двойственности является то, что *поразрядная сортировка обычно хуже сортировки методом слияния.* Это связано с тем, что метод выбора с замещением по сравнению с сортировкой методом слияния имеет определенное преимущество: нет очевидного пути такого построения поразрядной сортировки, чтобы можно было использовать процедуры внутренней сортировки, требующие более одной загрузки в память за один раз. На самом деле осциллирующая поразрядная сортировка часто будет порождать подфайлы, несколько меньшие, чем объем памяти, так что ее схема распределения соответствует дереву со значительно большим числом внешних узлов, чем было бы при использовании слияния и выбора с замещением. Соответственно возрастает длина внешнего пути дерева (т. е. время сортировки). (См. упр. 5.3.1–33.)

Для внешней поразрядной сортировки существует, однако, одно важное применение. Предположим, например, что имеется файл, содержащий фамилии всех сотрудников большого предприятия в алфавитном порядке. Предприятие состоит из десяти отделений, и требуется рассортировать файл по отделениям, *сохраняя* алфавитный порядок сотрудников каждого отделения. Если файл длинный, значит, сложилась именно та ситуация, в которой следует применять стабильную поразрядную сортировку, так как число записей, принадлежащих каждому из отделений, будет, вероятно, больше числа записей, которые были бы в начальных сериях, полученных методом выбора с замещением. Вообще говоря, если диапазон ключей так мал, что набор записей с одинаковыми ключами более чем вдвое превышает объем оперативной памяти, то разумно использовать поразрядную сортировку.

Мы видели в разделе 5.2.5, что на некоторых высокоскоростных компьютерах *внутренняя* поразрядная сортировка предпочтительнее слияния, поскольку "внутренний цикл" поразрядной сортировки обходится без сложных переходов. Если внешняя память очень быстрая, то для таких машин может оказаться проблемой выполнение слияния данных с такой скоростью, чтобы поспеть за оборудованием ввода-вывода. Поэтому в подобной ситуации поразрядная сортировка, возможно, лучше слияния, особенно если известно, что ключи распределены равномерно.

УПРАЖНЕНИЯ

1. [*20*] В начале раздела 5.4 было определено общее сбалансированное слияние на T лентах с параметром P, $1 \le P < T$. Покажите, что оно соответствует поразрядной сортировке, использующей систему счисления со смешанным основанием.

2. [*M28*] В тексте раздела схематически представлена многофазная поразрядная сортировка 21 ключа. Обобщите ее для F_n ключей и объясните, какие ключи и на какой ленте оказываются в конце каждой фазы. [*Указание.* Рассмотрите систему счисления, использующую числа Фибоначчи; см. упр. 1.2.8–34.]

3. [*M35*] Распространите результаты упр. 2 на многофазную поразрядную сортировку с четырьмя или более лентами (см. упр. 5.4.2–10.)

4. [*M23*] Докажите, что схема распределения Эшенхерста служит наилучшим способом сортировки десяти ключей на четырех лентах без обратного чтения в том смысле, что соответствующее дерево имеет минимальную длину внешнего пути среди всех "сильных 4-fifo-деревьев". (Таким образом, это, по существу, — наилучший метод, если не учитывать время перемотки.)

5. [*15*] Нарисуйте 4-lifo-дерево, соответствующее поразрядной сортировке Мочли с обратным чтением десяти ключей.

▶ **6.** [*20*] В некотором файле содержатся двухразрядные ключи 00, 01, ..., 99. После выполнения поразрядной сортировки Мочли по разряду единиц мы можем повторить ту же схему по разряду десятков, поменяв ролями ленты T2 и T4. В каком порядке, в конце концов, расположатся ключи на T2?

7. [*21*] Применим ли принцип двойственности также к файлам на нескольких бобинах?

*5.4.8. Сортировка с двумя лентами

Для того чтобы при выполнении слияния не происходило чрезмерного движения лент, необходимо иметь три НМЛ. Интересно подумать о том, как можно рационально организовать внешнюю сортировку с использованием только двух лент. В 1956 году Г. Б. Демут (H. B. Demuth) предложил метод, представляющий собой комбинацию выбора с замещением и сортировки методом пузырька. Предположим, что исходные данные занимают ленту T1, и начнем с того, что считаем в память $P+1$ записей. Теперь выведем запись с наименьшим ключом на ленту T2 и заменим ее следующей исходной записью. Продолжаем выводить записи, для которых значение ключа наименьшее среди всех хранящихся в памяти в текущий момент, сохраняя дерево выбора или приоритетную очередь из $P+1$ элементов. Когда ввод, наконец, исчерпается, в памяти окажется P наибольших ключей файла; выведем их в порядке возрастания. Теперь перемотаем обе ленты и повторим этот процесс, выполняя считывание с T2 и запись на T1. При каждом таком проходе на свои места помещается еще, по крайней мере, P записей. В программу можно встроить простую проверку для определения момента, когда весь файл станет упорядоченным. Потребуется не более $\lceil (N-1)/P \rceil$ проходов.

Задумавшись на минутку, придем к выводу, что каждый проход этой процедуры эквивалентен P последовательным проходам сортировки методом пузырька (алгоритм 5.2.2B)! Если элемент имеет P или более инверсий, то при вводе он окажется меньше всех элементов в дереве и поэтому будет немедленно выведен (таким образом, исчезает P инверсий). Если элемент имеет менее P инверсий, то он попадает в дерево выбора и выводится раньше всех бóльших ключей (потеряв, таким образом, все свои инверсии). Если $P = 1$, происходит то же самое, что и в методе пузырька, как следует из теоремы 5.2.2I.

Следовательно, общее число проходов будет равно $\lceil I/P \rceil$, где I — максимальное число инверсий любого элемента. Из результатов раздела 5.2.2 вытекает, что среднее значение I есть $N - \sqrt{\pi N/2} + 2/3 + O(1/\sqrt{N})$.

Если размер файла не слишком превосходит объем оперативной памяти или если файл первоначально почти упорядочен, то эта сортировка методом пузырька P-го порядка будет выполняться довольно быстро. В действительности ей можно отдать предпочтение даже в том случае, когда имеются дополнительные накопители на магнитных лентах, поскольку весь процесс сортировки может закончиться раньше, чем оператор успеет установить третью ленту! С другой стороны, сортировка таким методом довольно больших файлов со случайно расположенными элементами будет выполняться весьма медленно, так как время работы приблизительно пропорционально N^2.

Рассмотрим, как реализуется этот метод для 100,000 записей в примере из раздела 5.4.6. Необходимо разумно выбрать P, чтобы учесть междублочные промежутки при совмещении операций чтения и записи с вычислениями. Так как в примере предполагается, что каждая запись состоит из 100 символов, а 100,000 символов заполняют оперативную память, то у нас будет место для двух буферов ввода и двух буферов вывода размером B, если выбрать значения P и B, такие, что

$$100(P+1) + 4B = 100000. \tag{1}$$

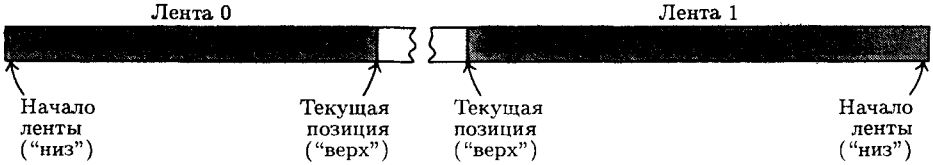
Если использовать обозначения, принятые в разделе 5.4.6, то приблизительное время выполнения каждого прохода выражается как

$$NC\omega\tau(1+\rho), \qquad \omega = (B+\gamma)/B. \tag{2}$$

Поскольку число проходов обратно пропорционально P, желательно выбрать такое B, кратное 100, которое минимизирует величину ω/P. Элементарный анализ показывает, что минимум достигается, когда B равно приблизительно $\sqrt{24975\gamma} + \gamma^2 - \gamma$. Поэтому мы выбираем $B = 3000$, $P = 879$. Положив в приведенных выше формулах $N = 100000$, получаем, что число проходов $\lceil I/P \rceil$ будет составлять около 114, а оценка общего времени решения — примерно 8.57 ч (предполагая для удобства, что исходные данные и окончательный выходной файл также имеют $B = 3000$). Здесь представлен случай, когда данные занимают около 0.44 бобины; для полной бобины потребовалось бы примерно в пять раз больше времени. Можно несколько улучшить показатели, предусмотрев в алгоритме периодические прерывания и пересылку записей с наибольшими ключами на вспомогательную ленту, которая затем снимается, поскольку эти записи просто копируются туда и обратно после того, как они уже были расположены в требуемом порядке.

Применение быстрой сортировки. Обменная сортировка с разделением, или быстрая сортировка (алгоритм 5.2.2Q), — это еще один метод внутренней сортировки, который предполагает почти последовательный просмотр данных. Можно ли изобрести нечто аналогичное для внешней сортировки на двух лентах? [N. B. Yoash, *CACM* **8** (1965), 649.]

Это несложно сделать, воспользовавшись обратным чтением. Предположим, что две ленты помечены 0 и 1, и представим, что файл располагается следующим образом.



Каждая лента играет роль стека. Две ленты, используемые, как представлено здесь, дают возможность считать файл линейным списком, в котором можно перемещать текущую позицию влево или вправо, копируя элементы из одного стека в другой. Следующие рекурсивные подпрограммы определяют соответствующую процедуру сортировки.

• SORT00 [рассортировать верхний подфайл с ленты 0 и вернуть его на ленту 0]. Если для подфайла достаточно места в оперативной памяти, то применить к нему внутреннюю сортировку и вернуть его на ленту. В противном случае выбрать одну запись R из подфайла; пусть ее ключом будет K. Читая ленту 0 в обратном направлении, копировать все записи, ключи которых $> K$, получая таким образом новый "верхний" подфайл на ленте 1. Теперь, считывая ленту 0 в прямом направлении, копировать все записи с ключами, равными K, на ленту 1. Затем, вновь читая ленту 0 в обратном направлении, копировать все записи с ключами $< K$ на ленту 1. Выполнив SORT10 над ключами $< K$, скопировать ключи, равные K, на ленту 0 и, наконец, выполнив SORT10 над ключами $> K$, завершить сортировку.

• SORT01 [рассортировать верхний подфайл с ленты 0 и записать его на ленту 1]. Процедура аналогична SORT00, но последнее обращение к "SORT10" заменено на "SORT11", за которым следует копирование ключей $\leq K$ на ленту 1.

• SORT10 [рассортировать верхний подфайл с ленты 1 и записать его на ленту 0]. Процедура такая же, как SORT01, но меняются местами 0 и 1, а также операторы отношений "<" и ">".

• SORT11 [рассортировать верхний подфайл с ленты 1 и вернуть его на ленту 1]. Процедура такая же, как SORT00, но меняются местами 0 и 1, а также отношения "<" и ">".

Можно без труда реализовать рекурсивное обращение к этим процедурам, записывая соответствующую управляющую информацию на ленты.

Если считать, что данные располагаются в случайном порядке и вероятность наличия равных ключей пренебрежимо мала, то можно оценить время выполнения этого алгоритма следующим образом. Пусть M — число записей, помещающихся в оперативной памяти. Пусть X_N — среднее число записей, считываемых при обращении к SORT00 или SORT11 применительно к подфайлу из N записей, где $N > M$, и пусть Y_N — соответствующая величина для SORT01 и SORT10. Тогда имеем:

$$\begin{aligned} X_N &= \begin{cases} 0, & \text{если } N \leq M; \\ 3N + 1 + \frac{1}{N}\sum_{0\leq k<N}(Y_k + Y_{N-1-k}), & \text{если } N > M; \end{cases} \\ Y_N &= \begin{cases} 0, & \text{если } N \leq M; \\ 3N + 2 + \frac{1}{N}\sum_{0\leq k<N}(Y_k + X_{N-1-k} + k), & \text{если } N > M. \end{cases} \end{aligned} \tag{3}$$

Решение этих рекуррентных соотношений (см. упр. 2) показывает, что общий объем информации, считываемой с ленты в течение фаз внешнего разделения, в среднем равен $6\frac{2}{3}N\ln N + O(N)$ при $N \to \infty$. Мы также знаем из формулы 5.2.2–(25), что среднее число фаз внутренней сортировки будет равно $2(N+1)/(M+2) - 1$.

Если применить этот анализ к примеру со 100,000 записями, рассмотренному в разделе 5.4.6, причем полагая, что в нашем распоряжении имеются буфера по 25,000 символов и время сортировки подфайла из $n \le M = 1000$ записей равно $2nC\omega\tau$, то получится среднее время сортировки, приблизительно равное 103 мин (включая, как на диаграмме А, окончательную перемотку). Итак, метод быстрой сортировки в среднем не плох, но, конечно, в *наихудшем* случае он ужасен и уступает даже методу пузырька, обсуждавшемуся выше. Тем не менее рандомизация сделает наихудший случай весьма маловероятным.

Поразрядная сортировка. Обменную поразрядную сортировку (алгоритм 5.2.2R) можно аналогичным образом приспособить для сортировки с двумя лентами, так как этот метод очень похож на быструю сортировку. В качестве трюка, который позволяет применить оба эти метода, можно воспользоваться идеей многократного чтения файла — тем, чего мы никогда не делали в предыдущих алгоритмах для работы с лентами.

Тот же трюк позволяет осуществить обычную поразрядную сортировку на двух лентах "сначала по младшей цифре". Имея исходные данные на T1, копируем на T2 все записи, ключ которых в двоичной системе оканчивается нулем. Затем, после перемотки T1, читаем ее вновь, копируя записи с ключами, оканчивающимися единицей. Теперь перематываются обе ленты и выполняется аналогичная пара проходов, но с заменой T1 лентой T2 и использованием *предпоследней* двоичной цифры. В этот момент на T1 будут содержаться все записи с ключами $(\ldots 00)_2$, за которыми следуют записи с ключами $(\ldots 01)_2$, затем — $(\ldots 10)_2$, затем — $(\ldots 11)_2$. Если ключи имеют размер b бит, то, чтобы завершить сортировку, потребуется только $2b$ проходов по всему файлу.

Подобную поразрядную сортировку можно применять только к *старшим* b бит ключа для некоторого разумно выбранного числа b; таким образом, если ключи были равномерно распределены, число инверсий уменьшится примерно в 2^b раз. После этого несколько проходов P-путевой сортировки методом пузырька позволят завершить работу.

Новый, но несколько более сложный подход к распределяющей сортировке с двумя лентами предложили А. И. Никитин и Л. И. Шолмов [*Кибернетика* **2**, 6 (1966), 79–84]. Имеются счетчики числа ключей (по одному на каждую возможную конфигурацию старших битов), на основе которых строятся искусственные ключи $\kappa_1, \kappa_2, \ldots, \kappa_M$ так, чтобы для каждого i число действительных ключей, лежащих между κ_i и κ_{i+1}, находилось в заранее заданном диапазоне между P_1 и P_2. Таким образом, M лежит между $\lceil N/P_2 \rceil$ и $\lceil N/P_1 \rceil$. Если счетчики старших битов не дают достаточной информации для определения таких $\kappa_1, \kappa_2, \ldots, \kappa_M$, то делается еще один или более проходов для подсчета частоты конфигураций менее значащих битов при некоторых конфигурациях старших битов. После того как таблица искусственных ключей построена, $2\lceil \lg M \rceil$ проходов будет достаточно для завершения сортировки. (Данный метод требует объема оперативной памяти, пропорционального N, и поэтому не может использоваться для внешней сортировки при $N \to \infty$.

На практике мы не станем применять его для файлов на нескольких бобинах, и, следовательно, M будет сравнительно невелико, так что таблица искусственных ключей легко поместится в памяти.)

Имитация дополнительных лент. Ф. К. Хенни (F. C. Hennie) и Р. Э. Стирнз (R. E. Stearns) изобрели общий метод имитации k лент всего на двух лентах, причем таким образом, что требуемое суммарное перемещение ленты возрастает всего лишь в $O(\log L)$ раз, где L — максимальное расстояние, которое нужно пройти на любой одной ленте [*JACM* **13** (1966), 533–546]. Их построение в случае сортировки можно слегка упростить, что и сделано в следующем методе, предложенном Р. М. Карпом.

Будем имитировать обычное сбалансированное слияние на четырех лентах, используя две ленты: T1 и T2. На первой из них (т. е. на T1) содержимое имитируемых лент хранится так, как изображено на рис. 86. Представим себе, что данные записаны на четырех "дорожках" по одной для каждой имитируемой ленты. (В действительности лента не имеет таких дорожек. Мы просто будем считывать блоки 1, 5, 9, 13, ... , как дорожку 1, блоки 2, 6, 10, 14, ... , как дорожку 2, и т. д.) Другая лента (T2) используется только для вспомогательного хранения, чтобы помочь в выполнении перестановок на T1.

	Зона 0	Зона 1		Зона 2							Зона 3		
Дорожка 1	1	5	9	13	17	21	25	29	33	37	41	45	49
Дорожка 2	2	6	10	14	18	22	26	30	34	38	42	46	50
Дорожка 3	3	7	11	15	19	23	27	31	35	39	43	47	51
Дорожка 4	4	8	12	16	20	24	28	32	36	40	44	48	52

Рис. 86. Разбивка ленты T1 в конструкции Хенни-Стирнза (непустые зоны заштрихованы).

Блоки на каждой дорожке разделяются на *зоны*, содержащие соответственно 1, 2, 4, 8, ..., 2^k, ... блоков. Зона k на каждой дорожке либо заполнена точно 2^k блоками данных, либо пуста. Например, на рис. 86 на дорожке 1 данные содержатся в зонах 1 и 3, на дорожке 2 — в зонах 0, 1 и 2, на дорожке 3 — в зонах 0 и 2, на дорожке 4 — в зоне 1, а все остальные зоны пусты.

Предположим, что мы выполняем слияние данных с дорожек 1 и 2 на дорожку 3. В оперативной памяти находятся два буфера, используемые двухпутевым слиянием для ввода, а также третий буфер — для вывода. Когда буфер ввода для дорожки 1 станет пустым, можно заполнить его следующим образом: найти первую непустую зону дорожки 1, скажем зону k, и скопировать ее первый блок в буфер ввода, затем скопировать остальные $2^k - 1$ блоков данных на T2 и переместить их в зоны 0, 1, ..., $k-1$ дорожки 1. (Теперь зоны 0, 1, ..., $k-1$ заполнены, зона k пуста.) Аналогичная процедура используется, чтобы заполнить буфер ввода для дорожки 2, как только он станет пустым. Когда буфер вывода подготовлен для записи на дорожку 3, мы обращаем этот процесс, т. е. просматриваем T1, пока не найдется первая *пустая* зона на дорожке 3, скажем зона k, и в то же время копируем данные из зон 0, 1, ..., $k-1$ на T2. Данные на T2, к которым присоединяется содержимое буфера вывода, используются теперь для заполнения зоны k на дорожке 3.

Для этой процедуры необходимо уметь выполнять запись в середину ленты T1, не разрушая последующую информацию. Как и в случае осциллирующей сортировки с прямым чтением (раздел 5.4.5), можно без опасений выполнять это действие, если принять меры предосторожности.

Поскольку просмотр до зоны k осуществляется только один раз за каждые 2^k шагов, то, чтобы переписать $2^l - 1$ блоков с дорожки 1 в память, потребуется переместить ленту на $\sum_{0 \le k < l} 2^{l-1-k} \cdot c \cdot 2^k = cl2^{l-1}$, где c — некоторая константа. Таким образом, каждый проход слияния требует $O(N \log N)$ шагов. Поскольку в сбалансированном слиянии имеется $O(\log N)$ проходов, общее время работы будет составлять $O\big(N(\log N)^2\big)$, что асимптотически значительно лучше, чем наихудший случай для быстрой сортировки.

На самом же деле этот метод оказывается почти бесполезным, если применяется для сортировки 100,000 записей из примера раздела 5.4.6, поскольку информация, которая должна размещаться на ленте T1, не поместится на одной бобине ленты. И даже если мы пренебрежем этим фактом и будем исходить из самых оптимистических предположений относительно совмещения чтения/записи/вычислений, относительно длин междублочных промежутков и т. д., то найдем, что для выполнения сортировки потребуется около 37 ч! Итак, этот метод представляет чисто академический интерес: константа в $O\big(N(\log N)^2\big)$ слишком велика, чтобы метод можно было эффективно использовать при реальных значениях N.

Одноленточная сортировка. Можно ли обойтись всего одной лентой? Нетрудно видеть, что сортировку методом пузырька P-го порядка можно преобразовать в одноленточную сортировку, но результат будет ужасен. Г. Б. Демут [Ph. D. thesis (Stanford University, 1956), 85] сделал вывод, что в компьютере с ограниченной оперативной памятью после просмотра ограниченного участка ленты нельзя уменьшить число инверсий перестановки больше чем на ограниченную величину. Следовательно, любой алгоритм сортировки с одной лентой потребует в среднем не более чем $N^2 d$ единиц времени (где d — некоторая положительная константа, зависящая от конфигурации компьютера).

Р. М. Карп нашел очень интересный подход к исследованию этой задачи, обнаружив то, что, по существу, является *оптимальным* способом сортировки с одной лентой. При анализе алгоритма Карпа удобно следующим образом переформулировать задачу: *как быстрее всего перевезти людей между этажами, если работает только один лифт?* [См. *Combinatorial Algorithms*, edited by Randall Rustin (Algorithmics Press, 1972), 17–21.]

Рассмотрим здание с n этажами; помещение каждого этажа рассчитано на b человек. В этом здании нет ни окон, ни дверей, ни лестниц, но есть лифт, который может останавливаться на любом этаже. В здании находятся bn человек, и ровно b из них хотят попасть на свой этаж. В лифт вмещается самое большее m человек, и он затрачивает одну единицу времени для перемещения с этажа i на этаж $i + 1$. Как найти самый быстрый способ перемещения всех людей на нужные этажи, если требуется, чтобы лифт начал и закончил свое движение на первом этаже?

Нетрудно заметить связь между этой задачей и одноленточной сортировкой: люди — это записи, здание — лента, этажи — отдельные блоки на ленте, а лифт — оперативная память компьютера. Действиям программ для компьютера свойственна бо́льшая гибкость, чем действиям лифтера (программа может, напри-

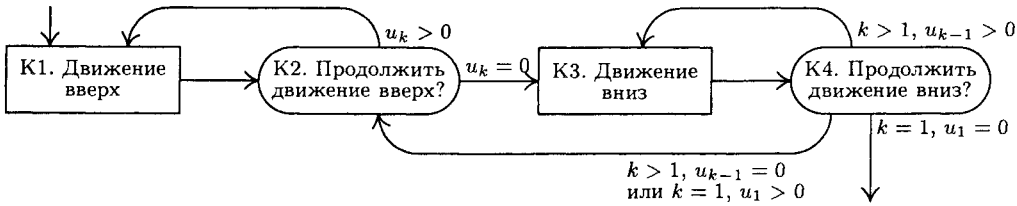


Рис. 87. Алгоритм Карпа для примера с лифтом.

мер, создавать двойников или, разрезав человека на две части, оставить их на время на разных этажах и т. д.), но в приведенном ниже алгоритме задача решается самым быстрым мыслимым способом без выполнения таких операций.

Алгоритм Карпа (рис. 87) использует два следующих вспомогательных массива:

$$\begin{aligned} &u_k,\ 1 \le k \le n\text{: число людей на этажах } \le k\text{, стремящихся попасть на этажи } > k;\\ &d_k,\ 1 \le k \le n\text{: число людей на этажах } \ge k\text{, стремящихся попасть на этажи } < k. \end{aligned} \tag{4}$$

Когда лифт пуст, мы всегда имеем $u_k = d_{k+1}$ при $1 \le k < n$, поскольку на каждом этаже находится b человек. Количество людей, направляющихся с этажей $\{1, \ldots, k\}$ на этажи $\{k+1, \ldots, n\}$, должно быть равно числу людей, стремящихся переехать в обратном направлении. По определению $u_n = d_1 = 0$.

Ясно, что лифт должен сделать, по крайней мере, $\lceil u_k/m \rceil$ рейсов с этажа k на этаж $k+1$ при $1 \le k < n$, так как только m пассажиров могут подняться за один рейс. Аналогично он должен сделать не менее $\lceil d_k/m \rceil$ рейсов с этажа k на этаж $k-1$. Следовательно, лифту потребуется, по крайней мере,

$$\sum_{k=1}^{n} \left(\lceil u_k/m \rceil + \lceil d_k/m \rceil \right) \tag{5}$$

единиц времени при любом правильном графике работы. Карп обнаружил, что эта нижняя граница действительно достижима, если $u_1, \ldots, u_{n-1}$ ненулевые.

Теорема К. *Если $u_k > 0$ при $1 \le k < n$, то существует график работы лифта, при котором все люди доставляются на свои этажи за минимальное время* (5).

Доказательство. Предположим, что в здании имеется дополнительно m человек. Изначально они находятся в лифте, и этаж их назначения искусственно полагается нулевым. Лифт может функционировать в соответствии со следующим алгоритмом, начиная с k (текущий этаж), равного 1.

K1. [Движение вверх.] Из $b+m$ людей, которые находятся в данный момент в лифте и на этаже k, только m человек, стремящихся на самый высокий этаж, попадает в лифт. Остальные остаются на этаже k.

Пусть теперь в лифте находится u человек, стремящихся на этажи $> k$, и d — на этажи $\le k$. (Если $u_k < m$, окажется, что $u = \min(m, u_k)$. Следовательно, можно увозить некоторых людей от их места назначения. Это — их жертва во имя общей пользы.) Уменьшить u_k на u, увеличить d_{k+1} на d и затем увеличить k на 1.

K2. [Продолжить движение вверх?] Если $u_k > 0$, вернуться к шагу K1.

K3. [Движение вниз.] Из $b + m$ людей, находящихся в данный момент в лифте или на этаже k, только m человек, стремящихся на самый низкий этаж, попадает в лифт. Остальные остаются на этаже k.

Пусть теперь в лифте находится u человек, стремящихся на этажи $\geq k$, и d — на этажи $< k$. (Всегда оказывается, что $u = 0$, а $d = m$, но алгоритм описывается здесь применительно к произвольным u и d, чтобы сделать доказательство несколько прозрачнее.) Уменьшить d_k на d, увеличить u_{k-1} на u и затем уменьшить k на 1.

K4. [Продолжить движение вниз?] Если $k > 1$ и $u_{k-1} > 0$, вернуться к шагу K3. Если $k = 1$ и $u_1 = 0$, закончить выполнение алгоритма (все люди доставлены на свои этажи, а m "дополнительных" человек снова находятся в лифте). В противном случае вернуться к шагу K2.

На рис. 88 показан пример выполнения этого алгоритма для здания с девятью этажами, $b = 2$ и $m = 3$. Заметим, что один из тех, кто стремится на шестой этаж, временно отдаляется от своего места назначения, несмотря на то что лифт проходит максимально близко к этому этажу. Идея проверки u_{k-1} на шаге K4 является, как мы увидим, решающим моментом для правильной работы алгоритма.

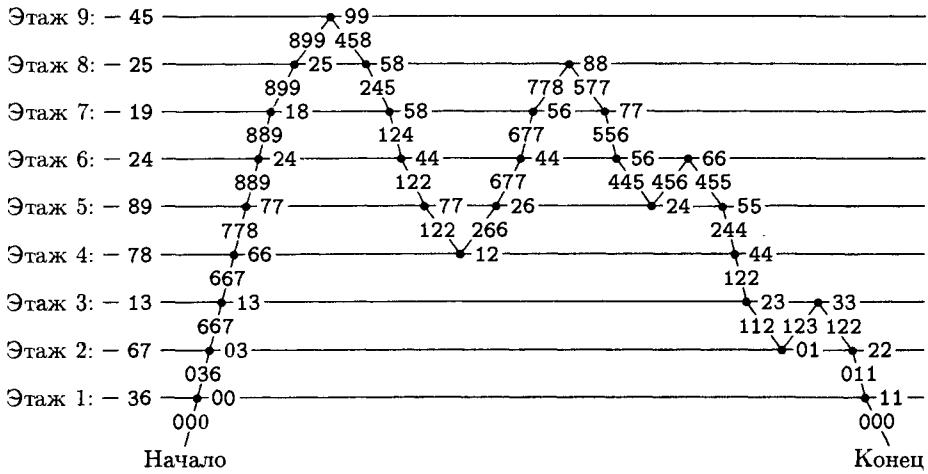


Рис. 88. Оптимальный способ перераспределения людей при помощи небольшого медленного лифта. (Каждый человек представлен номером этажа, на который он направляется.)

Чтобы проверить правильность этого алгоритма, заметим, что на шагах K1 и K3 всегда поддерживается соответствие данных в массивах u и d в (4) текущему состоянию, если считать людей в лифте находящимися на текущем этаже k. Теперь можно доказать по индукции, что в начале каждого шага справедливы следующие соотношения между параметрами:

$$u_l = d_{l+1} \qquad \text{при } k \leq l < n; \tag{6}$$

$$u_l = d_{l+1} - m \qquad \text{при } 1 \leq l < k; \tag{7}$$

$$u_{l+1} = 0, \qquad \text{если } u_l = 0 \text{ и } k \leq l < n. \tag{8}$$

Кроме того, в начале шага K1 в лифте или на этаже k находится $\min(u_k, m)$ человек, стремящихся на самые высокие этажи, среди всех людей на этажах $\le k$, которые хотят попасть на этажи $> k$. В начале шага K3 в лифте или на этаже k находится $\min(d_k, m)$ человек, стремящихся попасть на самые низкие этажи, среди всех людей на этажах $\ge k$, направляющихся на этажи $< k$. Эти условия также можно проверить по индукции, если проследить, как мы попадаем на шаг K1 или K3 (см. упр. 5).

Из сказанного выше следует, что замечания в скобках на шагах K1 и K3 справедливы. После каждого выполнения шага K1, следовательно, $\lceil u_k/m \rceil$ уменьшается на 1 и $\lceil d_{k+1}/m \rceil$ остается без изменений. После каждого выполнения шага K3 $\lceil d_k/m \rceil$ уменьшается на 1 и $\lceil u_{k-1}/m \rceil$ остается неизменным. Значит, алгоритм должен завершиться за конечное число шагов, и после этого в силу (6) и (8) каждый человек должен оказаться на своем этаже. ▌

Если $u_k = 0$, а $u_{k+1} > 0$, мы имеем "несвязную" ситуацию; лифт должен подняться до этажа $k+1$, чтобы переместить людей вверх, хотя никому не нужно переезжать с этажей $\le k$ на этажи $\ge k+1$. Не поступаясь общностью, можно считать $u_{n-1} > 0$. Тогда любой правильный график должен включать по крайней мере

$$2 \sum_{1 \le k < n} \max\bigl(1, \lceil u_k/m \rceil\bigr) \tag{9}$$

движений, так как мы требуем, чтобы лифт вернулся на первый этаж. График, для которого достигается эта нижняя граница, несложно построить (см. упр. 4).

УПРАЖНЕНИЯ

1. [*20*] В методе пузырька P-го порядка, который проанализирован в тексте раздела, используются только прямое чтение и перемотка. Можно ли модифицировать алгоритм так, чтобы получить преимущества от *обратного* чтения?

2. [*M26*] Найдите явные выражения в замкнутом виде для чисел X_N и Y_N, определенных в (3). [*Указание.* Проанализируйте решение уравнения 5.2.2–(19).]

3. [*38*] Существует ли метод сортировки с двумя лентами, основанный на сравнении ключей (а не на свойствах, представляющих эти ключи цифр), для которого в наихудшем случае при сортировке N записей перемещение лент составляет $O(N \log N)$? [При быстрой сортировке это значение достигается в среднем, но не в наихудшем случае; в методе Хенни-Стирнза (см. рис. 86) оно равняется $O(N(\log N)^2)$.]

4. [*M23*] В задаче о лифте предположим, что имеются индексы p и q, причем $q \ge p+2$, $u_p > 0$, $u_q > 0$ и $u_{p+1} = \cdots = u_{q-1} = 0$. Объясните, как составить график, требующий не более (9) единиц времени.

▶ **5.** [*M23*] Верно ли следующее утверждение? После шага K1 алгоритма теоремы K никто в лифте не стремится попасть на более низкий этаж, кроме тех, кто остался на этажах $< k$.

6. [*M30*] (Р. М. Карп.) Обобщите задачу о лифте (см. рис. 87) для случая, когда на этаже j первоначально находится b_j пассажиров и на этаж j стремится попасть b'_j пассажиров при $1 \le j \le n$. Покажите, что существует график работы, рассчитанный на $2\sum_{k=1}^{n-1} \max(1, \lceil u_k/m \rceil, \lceil d_{k+1}/m \rceil)$ единиц времени, причем на этаже j никогда не оказывается одновременно более $\max \max(b_j, b'_j)$ пассажиров. [*Указание.* Введите, если необходимо, фиктивных людей, чтобы равенство $b_j = b'_j$ соблюдалось при всех j.]

7. [*M40*] (Р. М. Карп.) Обобщите задачу из упр. 6, заменив линейный путь, который проходит лифт, сетью дорог, по которым можно ездить на автобусе, при условии, что

сеть образует любое *свободное дерево*. Автобус имеет конечную емкость, и желательно перевезти пассажиров к их местам назначения так, чтобы автобус прошел минимальное расстояние.

8. [*M32*] Пусть в задаче о лифте, рассмотренной в тексте раздела, $b = 1$. Сколько перестановок из n человек по n этажам дадут в (4) $u_k \le 1$ при $1 \le k \le n$? [Например, перестановка 3 1 4 5 9 2 6 8 7.]

▶ **9.** [*M25*] Найдите важную связь между шейкер-сортировкой, описанной в разделе 5.2.2 (см. рис. 16), и числами $u_1, u_2, \ldots, u_n$ в (4) для $b = 1$.

10. [*20*] Как бы вы сортировали файлы на нескольких бобинах, имея только два лентопротяжных устройства?

*5.4.9. Диски и барабаны

До сих пор ленты рассматривались как единственное средство для внешней сортировки, однако нередко в нашем распоряжении оказываются и другие типы более функционально гибких устройств внешней памяти. Хотя существует множество конструкций таких запоминающих устройств большого объема или запоминающих устройств с прямым доступом, можно выделить следующие общие для них свойства.

i) Получить доступ к любой заданной части хранимой информации можно довольно быстро.

ii) Блоки данных, содержащие последовательные слова, могут быстро передаваться между внутренней (оперативной) и внешней памятью.

Накопители на магнитной ленте обладают свойством (ii), но не (i), поскольку перемотка ленты от одного конца к другому отнимает много времени.

Каждое внешнее запоминающее устройство имеет свои особенности, которые следует тщательно изучить, прежде чем разрабатывать для него большие программы. Однако технология меняется так быстро, что здесь не удастся сколько-нибудь подробно обсудить все существующие разновидности оборудования. Поэтому мы рассмотрим лишь некоторые типичные внешние запоминающие устройства и на них проиллюстрируем продуктивные подходы к задаче сортировки.

Одним из наиболее распространенных типов внешних запоминающих устройств, удовлетворяющих (i) и (ii), является накопитель на магнитном диске (НМД), схематически показанный на рис. 89. Данные хранятся на нескольких быстро вращающихся круглых дисках, покрытых магнитным материалом. Для записи или выборки информации используется держатель головок в виде гребешка, включающий одну или несколько головок чтения/записи для каждой поверхности диска. Каждая поверхность делится на концентрические кольца, называемые *дорожками* или *треками*, так что за время одного оборота диска под головкой чтения/записи проходит целая дорожка. Держатель головок может перемещаться в двух направлениях — внутрь и наружу, передвигая головки чтения/записи от одной дорожки к другой, но это движение требует времени. Множество дорожек, которые могут быть прочитаны или записаны без перемещения держателя головок, называется *цилиндром*. Например, на рис. 89 показан НМД, который имеет по одной головке чтения/записи на каждую поверхность; пунктирными линиями обозначен один из цилиндров, состоящий из всех дорожек, просматриваемых в настоящий момент головками.

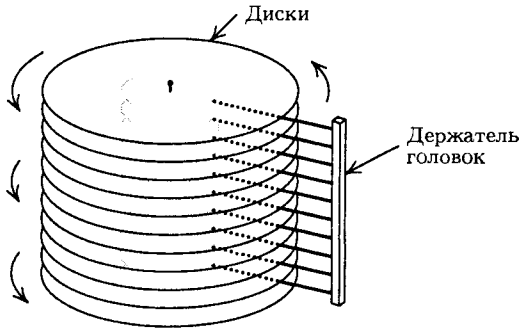


Рис. 89. Накопитель на магнитных дисках.

Чтобы конкретизировать эти рассуждения, рассмотрим гипотетический НМД MIXTEC, для которого

$$\begin{aligned} 1 \text{ дорожка} &= 5000 \text{ символов}, \\ 1 \text{ цилиндр} &= 20 \text{ дорожек}, \\ 1 \text{ дисковый накопитель} &= 200 \text{ цилиндров}. \end{aligned}$$

На таком устройстве хранится 20 млн символов, т. е. чуть меньше того объема данных, который можно записать на одну магнитную ленту в НМЛ MIXT. В некоторых устройствах на дорожках вблизи центра содержится меньше символов, чем на дорожках, расположенных ближе к краю. Это значительно усложняет программирование, но MIXTEC, к счастью, не создает подобных проблем. (См. раздел 5.4.6, в котором обсуждаются характеристики НМЛ MIXT. Здесь будут рассматриваться классические технологии, пригодные для работы с типичными для 70-х годов устройствами; более современные диски имеют бóльший объем хранимой информации и бóльшую скорость обращения к ней.)

Время, необходимое для чтения или записи на диск, представляет, по существу, сумму трех величин.

- Время поиска (время, затрачиваемое на перемещение держателя головок к нужному цилиндру).
- Время ожидания (задержка, связанная с вращением диска, пока головка чтения/записи не достигнет нужного места).
- Время передачи (задержка, связанная с вращением диска, пока данные проходят под головками).

В НМД MIXTEC время поиска для перехода от цилиндра i к цилиндру j равно $25 + \frac{1}{2}|i - j|$ мс. Если i и j — случайно выбранные целые числа между 1 и 200, то среднее значение $|i - j|$ равно $2\binom{201}{3}/200^2 \approx 66.7$, т. е. среднее время поиска составляет приблизительно 60 мс. Диски MIXTEC совершают один оборот за 25 мс, так что время ожидания равно в среднем 12.5 мс. Время передачи n символов равняется $(n/5000) \times 25$ мс $= 5n$ мкс. (Это приблизительно в $3\frac{1}{3}$ раза больше, чем скорость передачи для НМЛ MIXT, использованного в примерах из раздела 5.4.6.)

Таким образом, основные различия между НМД MIXTEC и НМЛ MIXT, касающиеся сортировки, следующие.

a) При работе с лентами возможен только последовательный доступ к данным.

b) Отдельная операция с диском, как правило, сопряжена со значительно бо́льшими накладными расходами (время поиска + время ожидания по сравнению со временем пуска и/или останова).

c) Скорость передачи у НМД выше.

Используя при работе с лентами разумные схемы слияния, можно до некоторой степени компенсировать недостаток (a). Теперь поставлена иная цель — найти такие рациональные алгоритмы сортировки на дисках, в которых компенсируется недостаток (b).

Способы сокращения времени ожидания. Рассмотрим сначала задачу минимизации задержек, причина которых заключается в том, что в момент, когда необходимо начать выполнение команды ввода-вывода, диск, как правило, не находится в подходящей позиции. Нельзя заставить диск вращаться быстрее, но все-таки можно прибегнуть к разным уловкам, которые уменьшат или даже полностью устранят время ожидания.

• Если мы считываем или записываем за один раз несколько дорожек одного цилиндра, то тем самым устраняем время ожидания (*и* время поиска) для всех дорожек, кроме первой. Вообще, зачастую можно таким образом синхронизировать вычисления с вращением диска, что при выполнении последовательности команд ввода-вывода не будет задержек из-за ожидания.

• Рассмотрим, как можно считать половину дорожки данных (рис. 90): если команда чтения выдается, когда головка находится на оси A, то задержка на ожидание отсутствует и общее время чтения равно времени передачи, т. е. $\frac{1}{2} \times 25$ мс. Если выполнение команды начинается, когда головка находится в точке B, то требуется $\frac{1}{4}$ оборота для ожидания и $\frac{1}{2}$ оборота для передачи; в итоге имеем $\frac{3}{4} \times 25$ мс. Наиболее интересен случай, когда головка первоначально находится в точке C: имея соответствующее оборудование и программное обеспечение, можно *не потерять* $\frac{3}{4}$ оборота на ожидание. Можно немедленно начать чтение во вторую половину буфера ввода, затем после паузы длительностью $\frac{1}{2} \times 25$ мс возобновить чтение в первую половину буфера, так что выполнение команды будет завершено, когда головка снова попадет в точку C. Поступая таким образом, можно гарантировать, что суммарное время ожидания и передачи никогда не превзойдет времени одного оборота независимо от начального положения диска. Среднее время ожидания уменьшается в результате с половины оборота до $\frac{1}{2}(1 - x^2)$ оборота, если читается или записывается доля x дорожки ($0 < x \leq 1$). Если читается или записывается целая дорожка ($x = 1$), этим методом можно *полностью* устранить ожидание.

Барабаны: случай, когда поиск не нужен. На некоторых устройствах внешней памяти установлено по одной головке чтения/записи для каждой дорожки, и поэтому время на поиск не отводится. Если на таком устройстве используется метод, продемонстрированный на рис. 90, то как время поиска, так и время ожидания сведены к нулю (при условии, что мы всегда читаем или записываем всю дорожку целиком). В этой идеальной ситуации время передачи является единственным ограничивающим фактором.

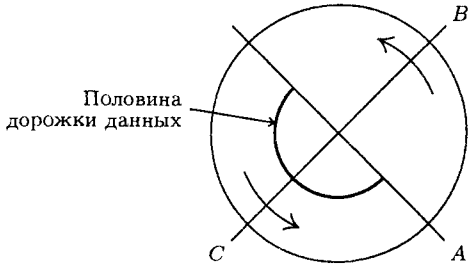


Рис. 90. Анализ времени ожидания при чтении половины дорожки.

Рассмотрим вновь пример из раздела 5.4.6, в котором сортируются 100,000 записей по 100 символов и используется оперативная память емкостью 100 000 символов. Весь объем сортируемых данных занимает половину диска MIXTEC. Обычно невозможно одновременно выполнять считывание и запись на одном дисковом устройстве, поэтому предположим, что имеется два диска и чтение и запись можно совместить. Пока будем считать, что диски — это фактически барабаны с 4 000 дорожек по 5 000 символов и никакого времени поиска не требуется.

Какой алгоритм сортировки следует использовать? Естественно выбрать метод слияния; другие методы внутренней сортировки не столь хороши в применении к дискам, за исключением, возможно, поразрядных методов, описанных в разделе 5.2.5. Но анализ, приведенный в разделе 5.4.7, показывает, что поразрядная сортировка обычно проигрывает слиянию в большинстве приложений общего назначения, поскольку теорема двойственности, сформулированная в этом разделе, применима к дискам точно так же, как и к лентам. Поразрядная сортировка действительно имеет преимущество в случае, когда ключи равномерно распределены и параллельно используется множество дисков, поскольку исходное распределение цифр ключей может помочь разделить всю процедуру на несколько независимых подпроцессов, никак не обменивающихся информацией. (См., например, R. C. Agarwal, *SIGMOD Record* **25**, 2 (June, 1996), 240–246.)

В дальнейшем основное внимание будет уделено сортировке методом слияния. Чтобы начать такую сортировку, можно использовать выбор с замещением с двумя буферами ввода по 5 000 символов и двумя буферами вывода по 5 000 символов. Фактически можно свести требования к оперативной памяти к *трем* буферам по 5 000 символов, если замещать записи в текущем буфере ввода записями, выводимыми из дерева выбора. Остается еще 85 000 символов (850 записей) для дерева выбора, так что один проход по данным, которые используются в этой книге в качестве примера, позволит сформировать около 60 начальных серий (см. формулу 5.4.6–(3)). Этот проход занимает лишь около 50 с, если предположить, что скорость внутренней обработки достаточно высока, чтобы успеть за вводом-выводом (каждые 500 мкс в буфер вывода должна передаваться запись). Если же сортируемые исходные данные находятся на НМЛ MIXT, а не на барабане, этот проход будет выполняться медленнее и время будет зависеть от скорости обмена данными с НМЛ.

Нетрудно видеть, что при использовании двух барабанов и при чтении/записи полных дорожек общее время передачи для P-путевого слияния уменьшается с уве-

личением P. К сожалению, нельзя выполнить одно только 60-путевое слияние всех начальных серий, так как в памяти нет места для 60 буферов. (Если размер буферов будет меньше 5 000 символов, то появится нежелательное время ожидания. Прошу не забывать, что мы все еще находимся в начале 70-х годов, когда возможности использования оперативной памяти были весьма ограничены.) Если выполнять P-путевое слияние, переписывая все данные с одного барабана на другой таким образом, что чтение и запись совмещены, то число проходов слияния равно $\lceil \log_P 60 \rceil$; следовательно, можно завершить работу за два прохода, если $8 \leq P \leq 59$. С уменьшением P уменьшается объем сопутствующих вычислений, поэтому выберем $P = 8$; если будет образовано 65 начальных серий, выберем $P = 9$. Если будет образовано 82 или больше начальных серий, можно взять $P = 10$, но, так как имеется место только для 18 буферов ввода и двух буферов вывода, могут возникнуть задержки в процессе слияния (см. алгоритм 5.4.6F). В таком случае, вероятно, лучше выполнить два частичных прохода, обрабатывая небольшую часть данных, и сократить таким образом число начальных серий до 81 или меньше.

При наших предположениях каждый проход слияния займет около 50 с, так что вся сортировка в этой идеальной ситуации будет завершена за 2.5 мин (плюс несколько секунд на инициализацию и другие вспомогательные операции). Это в шесть раз быстрее, чем при наилучшей из сортировок с шестью лентами, рассмотренных в разделе 5.4.6. Причинами такого ускорения являются увеличенная скорость передачи данных между внешней и оперативной памятью (она в 3.5 раза выше), более высокий порядок слияния (мы не можем осуществить 8-путевое слияние на лентах, не имея девяти или более лент) и то, что выводные данные остаются на диске (нет необходимости в заключительной перемотке и других аналогичных операциях). Если исходные данные и рассортированный результат должны находиться на НМЛ MIXT, а барабаны использоваться только для слияния, то понадобится около 8.2 мин.

Если имеется не два барабана, а только один, то время ввода-вывода увеличится вдвое, поскольку чтение и запись придется выполнять по отдельности. (На самом деле операции ввода-вывода займут в *три раза* больше времени, поскольку запись будет осуществляться поверх исходных данных. В таких случаях целесообразно за каждой операцией записи выполнять операцию контрольного чтения, чтобы не потерять необратимо какие-либо исходные данные, если только оборудование не обеспечивает автоматическую проверку записанной информации.) Однако часть этого дополнительного времени можно использовать эффективно, поскольку можно реализовать метод частичных проходов, при котором одни записи обрабатываются чаще других. Применение двух барабанов предполагает, что все данные обрабатываются четное или нечетное число раз, но в случае одного барабана можно использовать более общие схемы слияния.

В разделе 5.4.4 рассказывалось, что схемы слияния можно изображать с помощью деревьев и что время передачи, соответствующее схеме слияния, пропорционально длине внешнего пути дерева. В качестве схем эффективного слияния на лентах можно использовать лишь определенные виды деревьев (T-lifo или сильные T-fifo), потому что в процессе слияния некоторые серии оказываются "спрятанными" в середине ленты. Но при использовании *дисков или барабанов пригодны любые деревья*, если только степени их внутренних узлов не слишком велики (т. е. согласуются с действительным объемом внутренней памяти).

Следовательно, время передачи можно минимизировать, если выбрать дерево с минимальной длиной внешнего пути, такое, как полное P-арное дерево, где P — наибольшее возможное значение. По формуле 5.4.4–(9) длина внешнего пути такого дерева с S внешними узлами (листьями) равна

$$qS - \lfloor (P^q - S)/(P-1) \rfloor, \qquad q = \lceil \log_P S \rceil. \tag{1}$$

Особенно просто строится алгоритм, который осуществляет слияние в соответствии со схемой полного P-арного дерева. (См., например, рис. 91, на котором показан случай, когда $P = 3$, $S = 6$.) Сначала добавляем, если необходимо, фиктивные серии, чтобы сделать $S \equiv 1$ (по модулю $P-1$), затем объединяем серии в соответствии с правилом "первым включается — первым исключается", сливая на каждом этапе P самых "старых" серий в начале очереди в одну серию, помещаемую в конец.

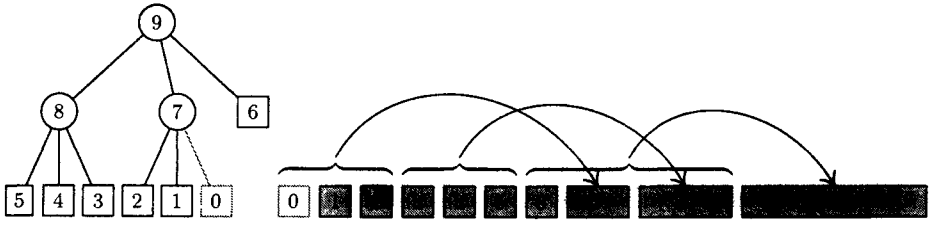


Рис. 91. Полное тернарное дерево с шестью листьями и соответствующая схема слияния.

Полные P-арные деревья дают оптимальную схему, если все серии имеют равную длину, но часто результат может быть еще лучше, если одни серии длиннее других. Можно без труда построить оптимальную схему для этой общей ситуации с помощью метода Хаффмэна (упр. 2.3.4.5–10), который применительно к слиянию формулируется так: "сначала добавьте $(1-S) \bmod (P-1)$ фиктивных серий длиной 0, затем многократно выполните слияние P *кратчайших* из имеющихся серий, пока не останется одна серия". Если все начальные серии имеют одинаковую длину, этот метод сводится к описанной выше дисциплине FIFO.

В нашем примере с обработкой 100,000 записей можно выполнять 9-путевое слияние, так как в памяти поместятся 18 буферов ввода и два буфера вывода и в алгоритме 5.4.6F будет достигнуто полное совмещение вычислений. Полное 9-арное дерево с 60 листьями соответствует схеме слияния с $1\frac{29}{30}$ прохода, если все начальные серии имеют одинаковую длину. Общее время сортировки с одним барабаном и с использованием контрольного чтения после каждой записи становится, таким образом, равным 7.4 мин. Увеличивая P, можно немного уменьшить это время, но ситуация будет весьма запутанной, поскольку не исключается задержка чтения в связи с тем, что буфера могут оказаться слишком полными или слишком пустыми.

Влияние времени поиска. Из всего вышесказанного следует, что для барабанов относительно легко построить оптимальную схему слияния, поскольку время поиска и время ожидания можно свести на нет. Но если используются диски, поиск информации отнимает больше времени, чем ее чтение. Поэтому время поиска оказывает значительное влияние на стратегию сортировки. Уменьшив порядок слияния P,

можно использовать бо́льшие по размеру буфера, так что поиск потребуется выполнять реже. За счет этого часто компенсируется дополнительное время передачи, которое растет с уменьшением P.

Время поиска зависит от расстояния, которое проходит держатель головок, и можно попытаться организовать работу таким образом, чтобы это расстояние было минимальным. Вероятно, разумно сначала сортировать записи внутри цилиндров. Однако довольно большое слияние требует большого количества переходов между цилиндрами (см., например, упр. 2). Кроме того, режим мультипрограммирования в современных операционных системах позволяет пользователю лишь в редких случаях по-настоящему контролировать положение держателя головок. Таким образом, предположение о том, что каждая команда для диска требует "случайного" поиска, почти всегда вполне оправдывается.

Наша цель в том и состоит, чтобы найти такое дерево (т. е. схему слияния), которое обеспечивает наилучший баланс между временем поиска и временем передачи. Для этого нужен некоторый способ, позволяющий оценить достоинства любого конкретного дерева по отношению к конкретной конфигурации оборудования. Рассмотрим, например, дерево на рис. 92. Необходимо оценить, сколько времени займет выполнение соответствующего слияния, чтобы можно было сравнить это дерево с другими.

Последующий анализ, который должен продемонстрировать некоторые общие идеи, будет выполнен в предположении, что (i) на чтение или запись n символов требуется $72.5 + 0.005n$ мс; (ii) под рабочее пространство в оперативной памяти отводится объем, достаточный для хранения 100 000 символов; (iii) для пересылки одного символа из буфера ввода в буфер вывода затрачивается в среднем 0.004 мс; (iv) *совмещение* чтения, записи и вычислений отсутствует; (v) размер буфера, используемого для вывода, необязательно равен размеру буфера, используемого для чтения данных на следующем проходе. Анализ задачи сортировки при этих простых предположениях будет полезен для понимания более сложных ситуаций.

Если выполняется P-путевое слияние, можно разделить внутреннюю рабочую память на $P + 1$ буферных областей: P — для ввода и 1 — для вывода; объем каждого буфера — $B = 100000/(P + 1)$ символов. Предположим, что в предназначенных для слияния файлах содержится в сумме L символов; значит, будет выполнено приблизительно L/B операций вывода и примерно столько же операций ввода. Следовательно, общее время слияния при таких предположениях будет равно (в миллисекундах) приблизительно

$$2\left(72.5\frac{L}{B} + 0.005L\right) + 0.004L = (0.00145P + 0.01545)L. \qquad (2)$$

Иными словами, для P-путевого слияния L символов необходимо примерно $(\alpha P + \beta)L$ машинных циклов, где α и β — некоторые константы, зависящие от времени поиска, времени ожидания, времени вычислений и объема памяти. Эта формула приводит к интересному способу построения хороших схем слияния для дисков. Рассмотрим, например, рис. 92 и будем считать, что все начальные серии (изображенные квадратными "листьями") имеют длину L_0. Тогда каждое слияние в узлах 9 и 10 выполняется за $(2\alpha + \beta)(2L_0)$ машинных циклов, слияние в узле 11 — за $(3\alpha + \beta)(4L_0)$ машинных циклов и окончательное слияние в узле 12 — за

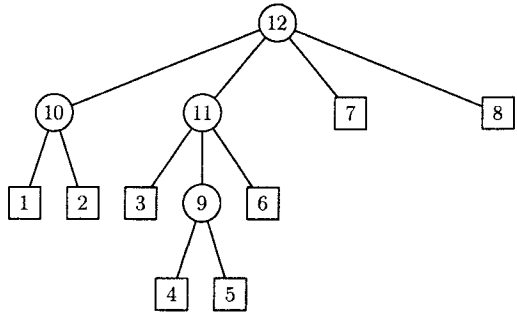


Рис. 92. Дерево с длиной внешнего пути 16 и длиной степенного пути 52.

$(4\alpha+\beta)(8L_0)$ машинных циклов. Общее время слияния, следовательно, составляет $(52\alpha+16\beta)L_0$ машинных циклов. Коэффициент 16 нам хорошо известен: это просто длина внешнего пути дерева. Коэффициент 52 при α соответствует новому понятию, которое можно назвать *длиной степенного пути дерева (degree path length)*. Эта длина равна взятой по всем листьям сумме степеней внутренних узлов, лежащих на пути от листа к корню. Например, на рис. 92 длина степенного пути равна $(2+4)+(2+4)+(3+4)+(2+3+4)+(2+3+4)+(3+4)+(4)+(4)=52$.

Если $\mathcal{T}$ — любое дерево, то пусть $D(\mathcal{T})$ и $E(\mathcal{T})$ обозначают соответственно длину степенного пути и длину внешнего пути этого дерева. Анализ сводится к следующей теореме.

Теорема H. *Если время, необходимое для выполнения P-путевого слияния L символов, составляет $(\alpha P+\beta)L$ и если требуется слить S серий равной длины, то наилучшая схема слияния соответствует дереву $\mathcal{T}$, для которого $\alpha D(\mathcal{T})+\beta E(\mathcal{T})$ минимально среди всех деревьев с S листьями.* ▌

(Эта теорема неявно содержалась в неопубликованной статье, которую Джордж А. Хаббард (George U. Hubbard) представил на Национальной конференции ACM в 1963 году.)

Пусть α и β — фиксированные константы. Будем говорить, что дерево *оптимально*, если оно имеет минимальное значение $\alpha D(\mathcal{T})+\beta E(\mathcal{T})$ среди всех деревьев $\mathcal{T}$ с тем же числом листьев. Нетрудно видеть, что *все поддеревья оптимального дерева также оптимальны*. Поэтому мы можем строить оптимальные деревья с n листьями, объединяя оптимальные деревья, у которых листьев меньше, чем n.

Теорема K. *Пусть последовательность чисел $A_m(n)$ определена при $1\le m\le n$ такими правилами:*

$$A_1(1)=0; \tag{3}$$

$$A_m(n)=\min_{1\le k\le n/m}\bigl(A_1(k)+A_{m-1}(n-k)\bigr) \qquad \text{при } 2\le m\le n; \tag{4}$$

$$A_1(n)=\min_{2\le m\le n}\bigl(\alpha mn+\beta n+A_m(n)\bigr) \qquad \text{при } n\ge 2. \tag{5}$$

Тогда $A_1(n)$ есть минимальное значение $\alpha D(\mathcal{T})+\beta E(\mathcal{T})$ среди всех деревьев $\mathcal{T}$ с n листьями.

Доказательство. Из соотношения (4) следует, что $A_m(n)$ — минимальное значение $A_1(n_1)+\cdots+A_1(n_m)$ для всех положительных чисел $n_1,\ldots,n_m$, таких, что $n_1+\cdots+n_m=n$. Требуемый результат получается теперь индукцией по n. ▌

Рекуррентные соотношения (3)-(5) можно использовать также для построения самих оптимальных деревьев. Пусть $k_m(n)$ — значение, для которого достигается минимум в определении $A_m(n)$. Тогда можно построить оптимальное дерево с n листьями, объединяя $m=k_1(n)$ поддеревьев в корне. Поддеревья являются оптимальными деревьями с $k_m(n)$, $k_{m-1}\bigl(n-k_m(n)\bigr)$, $k_{m-2}\bigl(n-k_m(n)-k_{m-1}(n-k_m(n))\bigr)$, ... листьями соответственно.

Эта конструкция при $\alpha=\beta=1$ проиллюстрирована в качестве примера в табл. 1. Краткие описания соответствующих оптимальных деревьев приводятся в правой части таблицы; элемент "4:9:9" для $n=22$, например, означает, что оптимальное дерево $\mathcal{T}_{22}$ с 22 листьями можно получить в результате объединения $\mathcal{T}_4$, $\mathcal{T}_9$ и $\mathcal{T}_9$ (рис. 93). Оптимальное дерево не является однозначной структурой; например, элемент 5:8:9 был бы столь же хорошим, как и 4:9:9.

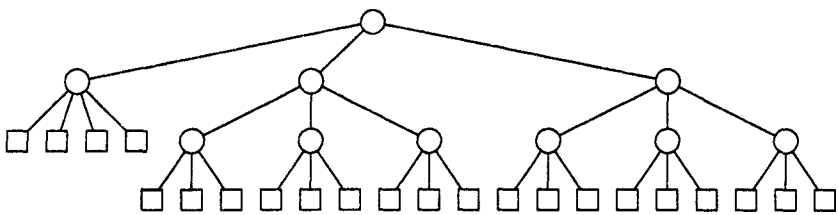


Рис. 93. Оптимальный способ слияния 22 начальных серий равной длины, если $\alpha=\beta$ в теореме Н. Эта схема позволяет минимизировать время поиска при предположениях, приведших к формуле (2).

Выражение (2) показывает, что всегда, когда используются $P+1$ буферов равных объемов, будет действительным соотношение $\alpha\le\beta$. Предельный случай, когда $\alpha=\beta$, показанный в табл. 1 и на рис. 93, возникает тогда, когда необходимо минимизировать само время поиска безотносительно ко времени передачи.

Вернемся к нашей первоначальной задаче. Мы еще не выяснили, как получить начальные серии на первом месте; без совмещения чтения/записи/вычислений метод выбора с замещением теряет некоторые из своих преимуществ. Возможно, следует заполнить всю оперативную память, рассортировать данные в ней и вывести результат. Каждую из таких операций ввода и вывода можно выполнить с одним поиском. Или, возможно, лучше использовать, скажем, 20% оперативной памяти в качестве комбинированного буфера ввода-вывода и выполнить выбор с замещением. Это требует в пять раз больше поисков (дополнительно примерно 60 с или около того!), но позволяет уменьшить число начальных серий со 100 до 64; уменьшение будет еще более резким, если исходный файл уже почти упорядочен.

Если не использовать выбор с замещением, то оптимальное дерево для $S=100$, $\alpha=0.00145$, $\beta=0.01545$ [см. (2)] оказывается весьма прозаическим. Это просто 10-путевое слияние, выполняемое за два прохода по данным. Выделив 30 с на внутреннюю сортировку (скажем, 100 быстрых сортировок), получаем, что начальный распределительный проход выполняется примерно за 2.5 мин, а каждый проход сли-

Таблица 1

ХАРАКТЕРИСТИКИ ОПТИМАЛЬНОГО ДЕРЕВА $A_m(n)$, $k_m(n)$, КОГДА $\alpha = \beta = 1$

n	m: 1	2	3	4	5	6	7	8	9	10	11	12	Дерево	n
1	0,0												—	1
2	6,2	0,1											1:1	2
3	12,3	6,1	0,1										1:1:1	3
4	20,4	12,1	6,1	0,1									1:1:1:1	4
5	30,5	18,2	12,1	6,1	0,1								1:1:1:1:1	5
6	42,2	24,3	18,1	12,1	6,1	0,1							3:3	6
7	52,3	32,3	24,1	18,1	12,1	6,1	0,1						1:3:3	7
8	62,3	40,4	30,2	24,1	18,1	12,1	6,1	0,1					2:3:3	8
9	72,3	50,4	36,3	30,1	24,1	18,1	12,1	6,1	0,1				3:3:3	9
10	84,3	60,5	44,3	36,1	30,1	24,1	18,1	12,1	6,1	0,1			3:3:4	10
11	96,3	72,4	52,3	42,2	36,1	30,1	24,1	18,1	12,1	6,1	0,1		3:4:4	11
12	108,3	82,4	60,4	48,3	42,1	36,1	30,1	24,1	18,1	12,1	6,1	0,1	4:4:4	12
13	121,4	92,4	70,4	56,3	48,1	42,1	36,1	30,1	24,1	18,1	12,1	6,1	3:3:3:4	13
14	134,4	102,5	80,4	64,3	54,2	48,1	42,1	36,1	30,1	24,1	18,1	12,1	3:3:4:4	14
15	147,4	114,5	90,4	72,3	60,3	54,1	48,1	42,1	36,1	30,1	24,1	18,1	3:4:4:4	15
16	160,4	124,7	102,4	80,4	68,3	60,1	54,1	48,1	42,1	36,1	30,1	24,1	4:4:4:4	16
17	175,4	134,8	112,4	90,4	76,3	66,2	60,1	54,1	48,1	42,1	36,1	30,1	4:4:4:5	17
18	190,4	144,9	122,4	100,4	84,3	72,3	66,1	60,1	54,1	48,1	42,1	36,1	4:4:5:5	18
19	205,4	156,9	132,5	110,4	92,3	80,3	72,1	66,1	60,1	54,1	48,1	42,1	4:5:5:5	19
20	220,4	168,9	144,4	120,5	100,4	88,3	78,2	72,1	66,1	60,1	54,1	48,1	5:5:5:5	20
21	236,5	180,9	154,4	132,4	110,4	96,3	84,3	78,1	72,1	66,1	60,1	54,1	4:4:4:4:5	21
22	252,3	192,10	164,4	142,4	120,4	104,3	92,3	84,1	78,1	72,1	66,1	60,1	4:9:9	22
23	266,3	204,11	174,5	152,4	130,4	112,3	100,3	90,2	84,1	78,1	72,1	66,1	5:9:9	23
24	282,3	216,12	186,5	162,5	140,4	120,4	108,3	96,3	90,1	84,1	78,1	72,1	5:9:10	24
25	296,3	229,12	196,7	174,4	150,5	130,4	116,3	104,3	96,1	90,1	84,1	78,1	7:9:9	25

яния — за почти 5 мин; в итоге имеем 12.4 мин. Если использовать выбор с замещением, то оптимальное дерево для $S = 64$ оказывается одинаково неинтересным (два прохода 8-путевого слияния); начальный распределительный проход осуществляется примерно за 3.5 мин, каждый проход слияния — примерно за 4.5 мин, а оценка общего времени составляет 12.6 мин. Не забудьте, что в обоих этих методах фактически полностью исключается совмещение чтения/записи/вычислений с тем, чтобы иметь бо́льшие буфера (для уменьшения времени поиска). Ни одна из этих оценок не включает время, которое может потребоваться для операций контрольного чтения.

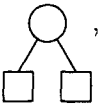
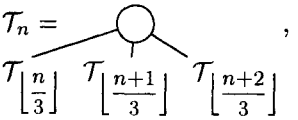
На практике последний проход слияния отличается от остальных; например, выводные данные часто редактируются и/или записываются на ленту. В таких случаях дерево, изображающее схему слияния, следует выбирать с использованием иного критерия оптимальности в корневом узле.

***Подробности об оптимальных деревьях.** В теоремах H и K интересно рассмотреть предельный случай, когда $\beta = 0$, несмотря на то что на практике обычно возникает соотношение между параметрами вида $0 \le \alpha \le \beta$. Какое дерево с n листьями имеет наименьшую возможную длину степенного пути? Любопытно, что в этом случае наилучшим оказывается 3-путевое слияние.

Теорема L. *Длина степенного пути дерева с n листьями никогда не будет меньше, чем*

$$f(n) = \begin{cases} 3qn + 2(n - 3^q), & \text{если } 2 \cdot 3^{q-1} \le n \le 3^q; \\ 3qn + 4(n - 3^q), & \text{если } 3^q \le n \le 2 \cdot 3^q. \end{cases} \tag{6}$$

Тернарные деревья $\mathcal{T}_n$, определенные правилами

$\mathcal{T}_1 = \square$, $\mathcal{T}_2 =$  , $\mathcal{T}_n =$  , (7)

имеют минимальную длину степенного пути.

Доказательство. Обратите внимание на то, что $f(n)$ — *выпуклая функция*, т. е.

$$f(n+1) - f(n) \geq f(n) - f(n-1) \qquad \text{при всех } n \geq 2. \tag{8}$$

Это свойство существенно в соответствии со следующей леммой, которая двойственна результату упр. 2.3.4.5–17.

Лемма C. *Функция $g(n)$, определенная на положительных целых числах, удовлетворяет условию*

$$\min_{1 \leq k < n} \bigl(g(k) + g(n-k)\bigr) = g\bigl(\lfloor n/2 \rfloor\bigr) + g\bigl(\lceil n/2 \rceil\bigr), \qquad n \geq 2, \tag{9}$$

тогда и только тогда, когда она выпуклая.

Доказательство. Если $g(n+1) - g(n) < g(n) - g(n-1)$ при некотором $n \geq 2$, то имеем $g(n+1) + g(n-1) < g(n) + g(n)$, что противоречит (9). Обратно, если (8) выполняется для g и если $1 \leq k < n-k$, то в силу выпуклости $g(k+1) + g(n-k-1) \leq g(k) + g(n-k)$. ▮

Последнюю часть доказательства леммы C можно обобщить для любого $m \geq 2$ и показать, что

$$\min_{\substack{n_1 + \cdots + n_m = n \\ n_1, \ldots, n_m \geq 1}} \bigl(g(n_1) + \cdots + g(n_m)\bigr)$$
$$= g\bigl(\lfloor n/m \rfloor\bigr) + g\bigl(\lfloor (n+1)/m \rfloor\bigr) + \cdots + g\bigl(\lfloor (n+m-1)/m \rfloor\bigr), \tag{10}$$

если g выпукла. Пусть

$$f_m(n) = f\bigl(\lfloor n/m \rfloor\bigr) + f\bigl(\lfloor (n+1)/m \rfloor\bigr) + \cdots + f\bigl(\lfloor (n+m-1)/m \rfloor\bigr). \tag{11}$$

Доказательство теоремы L будет полным, если убедиться, что $f_3(n) + 3n = f(n)$ и $f_m(n) + mn \geq f(n)$ при всех $m \geq 2$ (см. упр. 11). ▮

Было бы очень хорошо, если бы оптимальные деревья всегда характеризовались так же четко, как в теореме L. Но результаты для $\alpha = \beta$, которые приведены в табл. 1, показывают, что функция $A_1(n)$ не всегда выпукла. На самом деле данные, приведенные в табл. 1, достаточны, чтобы опровергнуть большинство простых предположений об оптимальных деревьях! Мы, однако, можем спасти часть теоремы L для общего случая. М. Шлюмбергер (M. Schlumberger) и Ж. Вуйлемен (J. Vuillemin) показали, что *высоких* порядков слияния всегда можно избежать.

Теорема M. *Если даны α и β, как в теореме H, то существует оптимальное дерево, в котором степень любого узла не превосходит*

$$d(\alpha, \beta) = \left\lceil \min_{k \geq 1} \left(k + \left(1 + \frac{1}{k}\right)\left(1 + \frac{\beta}{\alpha}\right)\right)\right\rceil. \tag{12}$$

Доказательство. Пусть $n_1, \ldots, n_m$ — положительные целые числа, такие, что $n_1 + \cdots + n_m = n$, $A(n_1) + \cdots + A(n_m) = A_m(n)$ и $n_1 \le \cdots \le n_m$, и предположим, что $m \ge d(\alpha, \beta) + 1$. Пусть k — значение, при котором достигается минимум в (12); покажем, что

$$\alpha n(m-k) + \beta n + A_{m-k}(n) \le \alpha n m + \beta n + A_m(n), \tag{13}$$

и, следовательно, минимум в (4) всегда достигается при некотором $m \le d(\alpha, \beta)$.

По определению, поскольку $m \ge k+2$, мы должны иметь

$$\begin{aligned} A_{m-k}(n) &\le A_1(n_1 + \cdots + n_{k+1}) + A_1(n_{k+2}) + \cdots + A_1(n_m) \\ &\le \alpha(n_1 + \cdots + n_{k+1})(k+1) + \beta(n_1 + \cdots + n_{k+1}) + A_1(n_1) + \cdots + A_1(n_m) \\ &= \big(\alpha(k+1) + \beta\big)(n_1 + \cdots + n_{k+1}) + A_m(n) \\ &\le \big(\alpha(k+1) + \beta\big)(k+1)n/m + A_m(n). \end{aligned}$$

Отсюда легко выводится (13). (Тщательный анализ этого доказательства показывает, что оценка (12) является "наилучшей из возможных" в том смысле, что некоторые оптимальные деревья должны иметь узлы степени $d(\alpha, \beta)$; см. упр. 13.) ▌

Для построения в теореме K требуется $O(N^2)$ ячеек памяти и $O(N^2 \log N)$ шагов для вычисления $A_m(n)$ при $1 \le m \le n \le N$; в теореме M показано, что необходимо только $O(N)$ ячеек и $O(N^2)$ шагов. Шлюмбергер и Вуйлемен открыли еще несколько очень интересных свойств оптимальных деревьев [*Acta Informatica* **3** (1973), 25–36]. Величина асимптотического выражения для $A_1(n)$ выводится, как продемонстрировано в упр. 9.

***Еще один способ распределения буферов.** В работе David E. Ferguson, *CACM* **14** (1971), 476–478, показано, что можно уменьшить время поиска, если не делать все буфера одного размера. Та же мысль и примерно в то же время пришла в голову еще нескольким авторам [S. J. Waters, *Comp. J.* **14** (1971), 109–112; Ewing S. Walker, *Software Age* **4** (August-September, 1970), 16–17].

Предположим, что мы выполняем 4-путевое слияние серий равной длины L_0, располагая оперативной памятью объемом M символов. Если разделить память на равные буфера размером $B = M/5$, то придется выполнить около L_0/B операций поиска для каждого входного файла и $4L_0/B$ операций поиска для выходного, что в сумме дает $8L_0/B = 40L_0/M$ операций поиска. Но если использовать четыре буфера ввода размером $M/6$ и один буфер вывода размером $M/3$, то потребуется всего лишь около $4 \times (6L_0/M) + 4 \times (3L_0/M) = 36L_0/M$ операций поиска! Время передачи в обоих случаях одно и то же, так что мы ничего не потеряем от этого изменения.

В общем случае предположим, что необходимо слить рассортированные файлы длиной $L_1, \ldots, L_P$ в рассортированный файл длиной $L_{P+1} = L_1 + \cdots + L_P$, и предположим, что для k-го файла используется буфер объемом B_k. Тогда

$$B_1 + \cdots + B_P + B_{P+1} = M, \tag{14}$$

где M — общий объем наличной оперативной памяти. Число операций поиска будет приблизительно равно

$$\frac{L_1}{B_1} + \cdots + \frac{L_P}{B_P} + \frac{L_{P+1}}{B_{P+1}}. \tag{15}$$

Попытаемся минимизировать эту величину при условии (14), считая для удобства, что B_k не обязаны быть целыми. Если увеличить B_j на δ и уменьшить B_k на ту же небольшую величину, то число поисков изменится на

$$\frac{L_j}{B_j+\delta}-\frac{L_j}{B_j}+\frac{L_k}{B_k-\delta}-\frac{L_k}{B_k}=\left(\frac{L_k}{B_k(B_k-\delta)}-\frac{L_j}{B_j(B_j+\delta)}\right)\delta,$$

т. е. сумму (15) можно уменьшить, если $L_j/B_j^2 \neq L_k/B_k^2$. Следовательно, мы получим минимальное число операций поиска только при таком условии:

$$\frac{L_1}{B_1^2}=\cdots=\frac{L_P}{B_P^2}=\frac{L_{P+1}}{B_{P+1}^2}. \tag{16}$$

Поскольку минимум обязательно существует, он должен достигаться при

$$B_k=\sqrt{L_k}\,M/\left(\sqrt{L_1}+\cdots+\sqrt{L_{P+1}}\right), \qquad 1\le k\le P+1 \tag{17}$$

(это единственные значения $B_1,\ldots,B_{P+1}$, удовлетворяющие одновременно (14) и (16)). Подставив (17) в (15), можно получить весьма простую формулу для общего числа операций поиска,

$$\left(\sqrt{L_1}+\cdots+\sqrt{L_{P+1}}\right)^2/M, \tag{18}$$

которую можно сравнить с результатом $(P+1)(L_1+\cdots+L_{P+1})/M$, получающимся в том случае, если длины всех буферов равны. Как следует из упр. 1.2.3–31, выигрыш равен

$$\sum_{1\le j<k\le P+1}\left(\sqrt{L_j}-\sqrt{L_k}\right)^2/M.$$

К сожалению, формула (18) не годится для простого определения оптимальной схемы слияния, как в теореме K (см. упр. 14).

Использование сцепления. В работе M. A. Goetz, *CACM* **6** (1963), 245–248, предложен интересный способ связывания отдельных дорожек, благодаря чему исключается поиск при выводе. Для реализации такой идеи требуется почти невообразимый набор программ, управляющих дисками; сама идея применима не только для сортировки, но и для решения многих других задач. Поэтому она может оказаться весьма полезной для выполнения задач общего назначения.

Идея проста. Вместо того, чтобы разместить дорожки последовательно внутри цилиндров диска, свяжем их вместе и будем хранить списки свободного пространства по одному для каждого цилиндра. Когда наступит время вывода дорожки информации, запишем ее на текущий цилиндр (на тот, на котором оказался держатель головок), если он не полон. При этом время поиска обычно сводится на нет.

Здесь нас подстерегает ловушка. Дело в том, что мы не можем записывать ссылку на следующую дорожку внутри самой дорожки, поскольку необходимая для этого информация в нужный момент неизвестна. (Можно, если это нас устраивает, записать ссылку на предыдущую дорожку и на следующем проходе читать файл в обратном направлении.) Допускается хранение отдельной таблицы адресов связи для дорожек каждого файла, возможно, частично на самом диске. Списки свободного пространства можно компактно представить с помощью битовых таблиц, в которых 1 000 бит указывает занятость или незанятость 1 000 дорожек.

Пересмотренный вариант прогнозирования. В алгоритме 5.4.6F показано, как можно предсказать, какой из входных буферов при P-путевом слиянии освободится первым. Для этого анализируется последний ключ в каждом буфере. В результате можно выполнять чтение и вычисление одновременно. В этом алгоритме используются плавающие входные буфера, не связанные жестко ни с одним из файлов. Таким образом, все буфера могут быть одного размера и можно не использовать каких-либо ухищрений при распределении оперативной памяти для буферов. Ограничения, вытекающие из использования буферов одинакового размера, в настоящее время несущественны, поскольку объем оперативной памяти в современных компьютерах не идет ни в какое сравнение с прежними объемами. Теперь само собой разумеющимся считается выделение буфера объемом в одну дорожку диска.

Таким образом, можно считать, что P серий, которые нужно слить, состоят из последовательности *блоков* данных, в которой каждый блок (кроме разве что последнего) содержит ровно B записей. Д. Л. Уитлоу (D. L. Whitlow) и А. Сассон (A. Sasson) разработали интересный алгоритм, названный ими SyncSort [*U. S. Patent 4210961* (1980)], который представляет собой усовершенстованный вариант алгоритма 5.4.6F и требует использования всего лишь трех буферов размером B вместе со специальным системным буфером (пулом), в котором можно хранить до PB записей и PB указателей. В отличие от этого нового алгоритма, алгоритм 5.4.6F требовал $2P$ входных буферов и 2 выходных, но не предполагал буферизации указателей.

Три буфера для SyncSort объединяются в кольцо. В процессе слияния компьютер обрабатывает данные в текущем буфере, в то время как входные данные считываются в следующий буфер, а выходные записываются в файл из третьего. Многие модели НМД позволяют записывать и считывать информацию одновременно при обращении к одному и тому же цилиндру. Таким образом, при слиянии можно записывать новые серии на место тех серий, которые считываются. Альтернативный вариант — использовать два диска. Тогда с одного из них выполняется чтение, а на другой производится запись. Если нельзя четко синхронизировать чтение и запись (например, из-за расхождений во времени поиска дорожки), можно включить в кольцо четыре или даже более буферов, как показано на рис. 26 в разделе 1.4.4.

Выполнение алгоритма SyncSort начинается с чтения первого блока каждой серии и передачи этих PB записей в системный буфер. Каждая запись в системном буфере связывается с последующей в той серии, которой она принадлежит, кроме последней, у которой еще нет последующей. Наименьший из ключей в последних записях определяет серию, которая первой нуждается в дальнейшей обработке, и мы начинаем считывать второй блок именно этой серии. Как только второй блок будет считан, начнется слияние. По значению заключительного ключа в этом блоке можно точно определить следующий подходящий блок и продолжить таким же способом точно угадывать очередность ввода блоков из исходного файла еще до того, как возникнет необходимость в их обработке.

Алгоритм слияния SyncSort предполагает обмен каждой записи в текущем буфере со следующей записью на выход, а именно — с записью в системном буфере, имеющей наименьший ключ. Дерево выбора и связи со следующими записями также обновляются по мере необходимости в процессе такого обмена. Как только будет

достигнут конец в текущем буфере, можно "провернуть" кольцо буферов — и буфер считывания станет текущим, буфер записи станет буфером для чтения, а данные из прежнего текущего буфера будут переписываться в выходной файл.

Использование нескольких дисков. Первые накопители на магнитных дисках были устройствами внушительных габаритов и веса, но со временем они стали намного меньше, легче, а главное — дешевле, но емкость их намного возросла. Это совершенно естественно привело к тому, что многие разработчики обратили внимание на создание специальных алгоритмов для ранее немыслимых комплексов из 5, 10 или даже 50 совместно работающих НМД.

Один из способов использования множества дисков — применение технологии *разделения данных (disk striping)* для больших файлов. Предположим, в нашем распоряжении имеется D НМД, пронумерованных 0, 1, ..., $D-1$, и рассмотрим файл, который состоит из L блоков $a_0 a_1 \ldots a_{L-1}$. Разделение этого файла на D дисков означает, что блок a_j записывается на диск под номером $j \bmod D$. Следовательно, на диске 0 хранятся блоки $a_0 a_D a_{2D} \ldots$, на диске 1 — $a_1 a_{D+1} a_{2D+1} \ldots$ и т. д. При такой организации можно выполнять D операций чтения или записи одновременно в группах из D блоков $a_0 a_1 \ldots a_{D-1}$, $a_D a_{D+1} \ldots a_{2D-1}$, ..., которые называются *суперблоками*. Отдельные блоки в каждом суперблоке должны находиться на соответствующих цилиндрах различных дисков, что обеспечит одинаковое время поиска на каждом устройстве. Теперь мы сможем работать, как если бы в нашем распоряжении был один-единственный диск с этими блоками данных и буфера объемом DB, но операции ввода и вывода выполнялись в D раз быстрее.

Благодаря такой организации разделения данных получается довольно изящный вариант усовершенствования сортировки при выполнении 2-путевого слияния или (в общем случае) всегда, когда необходимо привести в соответствие записи с равными ключами в двух файлах, упорядоченных по ключам. Предположим, что блоки $a_0 a_1 a_2 \ldots$ первого файла разделены между D дисками, как описано выше, но блоки $b_0 b_1 b_2 \ldots$ другого файла разделены в обратном порядке, т. е. блок b_j хранится на диске $(D-1-j) \bmod D$. Например, если $D = 5$, то блоки a_j находятся соответственно на дисках 0, 1, 2, 3, 4, 0, 1, ... , а блоки b_j, $j \geq 0$, находятся на дисках 4, 3, 2, 1, 0, 4, 3, ... Пусть α_j — последний ключ в блоке a_j, а β_j — последний ключ в блоке b_j. Проанализировав блоки α и β, можно предсказать нужный порядок считывания данных из этих блоков. Например, эта последовательность может иметь вид

$$a_0 b_0 a_1 a_2 b_1 \;\; a_3 a_4 b_2 a_5 a_6 \;\; a_7 a_8 b_3 b_4 b_5 \;\; b_6 b_7 b_8 b_9 b_{10} \;\; \ldots .$$

При $D = 5$ блоки находятся соответственно на дисках

$$0\ 4\ 1\ 2\ 3 \;\; 3\ 4\ 2\ 0\ 1 \;\; 2\ 3\ 1\ 0\ 4 \;\; 3\ 2\ 1\ 0\ 4 \;\; \ldots$$

и, если считывать их по пять одновременно, можно осуществлять ввод безо всяких помех с дисков $\{0,4,1,2,3\}$, $\{3,4,2,0,1\}$, $\{2,3,1,0,4\}$, $\{3,2,1,0,4\}$, ... При этом никогда не возникнет конфликт вследствие попытки прочесть одновременно два блока с одного и того же устройства! В общем случае, имея D дисков, можно бесконфликтно считывать D блоков сразу, поскольку при некотором k первая группа будет иметь k блоков $a_0 \ldots a_{k-1}$ на дисках от 0 до $k-1$, а $D-k$ блоков $b_0 \ldots b_{D-k-1}$ — на дисках от $D-1$ до k. Далее можно продолжать в том же духе, но с дисками, номера которых сдвигаются циклически на k.

Этот трюк хорошо известен карточным фокусникам, которые называют его *принципом Гилбрета*. Трюк изобретен в 60-х годах Норманом Гилбретом (Norman Gilbreath) [см. Martin Gardner, *Mathematical Magic Show* (New York: Knopf, 1977), Chapter 7; N. Gilbreath, *Genii* **52** (1989), 743–744]. Для того чтобы решить, какой блок следует считывать следующим, нужна информация из блоков α и β, но она занимает только малую долю всего объема a и b и, следовательно, можно хранить ее в отдельных файлах. Отсюда следует, что можно обойтись меньшими буферами для хранения считанных на полной скорости данных (см. упр. 23).

Рандомизированное разделение. Если нужно выполнить P-путевое слияние на D дисках при условии, что P и D достаточно велики, одновременное чтение информации с D дисков без конфликтов реализовать не удастся (если только не организовать большое количество буферов). Это связано с тем, что аналога принципу Гилбрета при $P > 2$ не существует. Как бы мы ни организовали распределение блоков файла между дисками, всегда есть вероятность, что придется считывать множество блоков в оперативную память прежде, чем появится возможность их использовать, поскольку те блоки, которые действительно нужно обрабатывать, могут оказаться на одном и том же диске.

Предположим, что нужно выполнить 8-путевое слияние на 5 дисках и блоки $a_0a_1a_2\ldots$, $b_0b_1b_2\ldots$, $\ldots$, $h_0h_1h_2\ldots$ по 8 серий разделены по такому правилу: a_j — на диск $j \bmod D$, b_j — на диск $(j+1) \bmod D$, $\ldots$, h_j — на диск $(j+7) \bmod D$. Необходимо реализовать доступ к этим блокам в следующем порядке:

$$a_0b_0c_0d_0e_0\ f_0g_0h_0d_1e_1\ d_2e_2d_3a_1f_1\ b_1g_1a_2f_2e_3\ d_4c_1h_1b_2g_2\ a_3f_3e_4d_5d_6\ \ldots. \tag{19}$$

Тогда они появятся на соответствующих дисках

$$0\ 1\ 2\ 3\ 4\ 0\ 1\ 2\ 4\ 0\ 0\ 1\ 1\ 1\ 1\ 2\ 2\ 2\ 2\ 2\ 2\ 3\ 3\ 3\ 3\ 3\ 3\ 3\ 3\ 3\ 4\ \ldots, \tag{20}$$

так что наилучший вариант для нас — вводить их следующим образом.

Такт 1	Такт 2	Такт 3	Такт 4	Такт 5
$a_0b_0c_0d_0e_0$	$f_0g_0h_0c_1d_1$	$e_1e_2b_1h_1d_6$	$d_2d_3g_1b_2$?	? $a_1a_2g_2$?

Такт 6	Такт 7	Такт 8	Такт 9
? $f_1f_2a_3$?	? ? e_3f_3 ?	? ? d_4e_4 ?	? ? ? d_5 ?

(21)

К тому моменту, когда понадобится использовать блок d_5, придется уже прочесть d_6 и еще 15 блоков следующих за ним данных, обозначенных через "?", вследствие переполнения на диске 3. А это нельзя сделать, не организовав семь буферов, в которых содержатся остатки от a_3, b_2, c_1, e_4, f_3, g_2 и b_1. Таким образом, в нашем примере потребуется пространство в памяти для, по крайней мере, $(16+8+5)B$ входных записей.

Вместо этого при использовании обычной процедуры распределения суперблоков при разделении дисков будем считывать блоки $a_0a_1a_2a_3a_4$ на такте 1, $b_0b_1b_2b_3b_4$ — на такте 2, $\ldots$, $h_0h_1h_2h_3h_4$ — на такте 8, $d_5d_6d_7d_8d_9$ — на такте 9 (поскольку следующим нам понадобится суперблок $d_5d_6d_7d_8d_9$) и т. д. Использование стратегии SyncSort потребует буферов для хранения $(P+3)DB$ записей и PDB указателей в оперативной памяти. Можно показать, что более динамичный подход, упомянутый выше, потребует не более половины этого объема буферов, но по-прежнему объем

необходимой оперативной памяти почти пропорционален PDB при больших P и D (см. упр. 24).

В работе R. D. Barve, E. F. Grove, J. S. Vitter, *Parallel Computing* **23** (1997), 601–631, показано, что небольшая модификация этого подхода с независимыми блоками позволяет получить алгоритм, который практически полностью использует скорость обмена данными с диском, но при этом требует буферов всего лишь для $O(P + D\log D)$ блоков вместо $\Omega(PD)$. Предложенная в работе технология *рандомизированного разделения (randomized striping)* предполагает помещение блока j серии k на диск $(x_k + j) \bmod D$, где x_k — случайное число, которое выбирается как раз перед тем, как будет впервые записана серия k. Вместо того чтобы постоянно использовать в качестве исходных данных D блоков по одному с каждого диска, применяется довольно простой механизм отслеживания наступления момента, когда не оказывается достаточного пространства для выполнения чтения вперед с определенных дисков. В приведенной выше работе доказано, что этот метод асимптотически стремится к оптимальному.

Для того чтобы реализовать P-путевое слияние на D дисках с рандомизированным разделением, нужно организовать $2D + P + Q - 1$ буферов плавающего ввода, каждый для одного блока из B записей. Исходные данные, как правило, считываются в D из этих буферов, называемых *активными буферами чтения*. В то же время P из оставшихся буферов содержат лидирующие блоки, записи из которых в текущий момент участвуют в слиянии. Такие буфера называются *активными буферами слияния*. Оставшиеся $D+Q-1$ прочих "разношерстных" буферов либо пусты, либо содержат данные, которые понадобятся позже. Неотрицательный параметр Q можно увеличить в процессе настройки системы, чтобы уменьшить вероятность возникновения ситуации, когда придется выполнять повторное чтение с любого из дисков.

Блоки всех серий можно скомпоновать в *хронологическом порядке*, как в (19): первыми идут блоки 0 каждой серии, за ними следуют другие, поддерживая при этом порядок, в котором освобождаются активные буфера слияния. Как было показано выше, такой порядок определен последними ключами в каждом блоке, так что несложно предсказать, какой из блоков может потребоваться первым.

Вновь рассмотрим пример (19), приняв $P = 8$, $D = 5$ и $Q = 4$. Теперь в нашем распоряжении имеется только $2D + P + Q - 1 = 21$ буферов для исходных блоков вместо 29, которые нужны были ранее для обеспечения максимальной скорости считывания. Будем использовать смещения

$$x_1 = 3,\ x_2 = 1,\ x_3 = 4,\ x_4 = 1,\ x_5 = 0,\ x_6 = 4,\ x_7 = 2,\ x_8 = 1 \tag{22}$$

(это цифры в числе π) для серий a, b, ..., h. Таким образом, если перечислить данные блоки в хронологическом порядке, на соответствующих дисках будет содержаться следующее.

Диск	Блоки																					
0:				e_0								f_1		a_2		d_4	c_1					...
1:	b_0		d_0			h_0		e_1							f_2					a_3		d_5 ...
2:					g_0		d_1		e_2			b_1						h_1			f_3	d_6 ...
3:	a_0								d_2				g_1		e_3				b_2			...
4:			c_0		f_0						d_3	a_1								g_2		e_4 ...

(23)

"Случайные" смещения в (22) в сочетании с последовательным разделением внутри каждой серии, будут стремиться к минимизации "заторов" в любой отдельной хронологической последовательности. В данном случае система будет работать следующим образом.

	Активные считывания	Активные слияния	Прочие	В ожидании
Цикл 1	$e_0 b_0 g_0 a_0 c_0$	$- - - - - - - - -$	$(- - - - - - - - -)$	a_0
Цикл 2	$f_1 d_0 d_1 d_2 f_0$	$a_0 - - - - - - - -$	$b_0 c_0 (e_0 g_0 - - - -)$	d_0
Цикл 3	$a_2 h_0 e_2 g_1 d_3$	$a_0 b_0 c_0 d_0 - - - -$	$e_0 f_0 g_0 (d_1 d_2 f_1 - -)$	h_0
Цикл 4	$a_2 e_1 b_1 g_1 a_1$	$a_0 b_0 c_0 d_0 e_0 f_0 g_0 h_0$	$d_1 (d_2 e_2 d_3 f_1 g_1 a_2 -)$	e_1
Цикл 5	$d_4 f_2 h_1 e_3 g_2$	$a_0 b_0 c_0 d_1 e_1 f_0 g_0 h_0$	$d_2 e_2 d_3 a_1 f_1 b_1 g_1 a_2 ()$	f_2
Цикл 7	$c_1 a_3 f_3 b_2 e_4$	$a_2 b_1 c_0 d_3 e_2 f_2 g_1 h_0$	$e_3 d_4 (h_1 g_2 - - - -)$	c_1
Цикл 7	$? \, d_5 d_6 \, ? \, ?$	$a_2 b_1 c_1 d_4 e_3 f_2 g_1 h_0$	$h_1 b_2 g_2 a_3 f_3 e_4 (- -)$	d_5

(24)

В каждом очередном цикле поджидается первый в хронологическом порядке блок, который не участвовал в слиянии и не находится в прочих буферах. Это один из блоков, которые считываются в текущий момент в активный буфер ввода. Предполагается, что компьютер работает значительно быстрее, чем НМД, и, таким образом, все блоки, находящиеся перед тем, который ожидается, уже задействованы в процессе слияния до завершения ввода. Предполагается также, что в нашем распоряжении имеется достаточный объем буферов и выполнение слияния не задерживается из-за отсутствия места для размещения результата (см. упр. 26). Когда завершится цикл ввода, блок, который мы ожидаем, немедленно классифицируется как принадлежащий активному буферу слияния, а опустевший буфер слияния, который, таким образом, выведен из этой категории, будет использоваться в составе активных буферов чтения. Другие $D-1$ активных буферов чтения выбираются среди $D-1$ наименее важных прочих буферов. Буфера этой последней группы ранжируются в хронологическом порядке по содержимому. В следующем цикле нам придется ожидать первый блок, еще не вовлеченный в слияние и не представленный в прочих буферах. Любой из прочих буферов, предшествующих этому блоку в хронологическом порядке, становится частью группы активных буферов слияния прежде, чем начнется очередной цикл ввода, но другие — показанные выше в скобках — сохранят свой статус прочего буфера в следующем цикле. Однако в следующий цикл с сохранением статуса может быть перенесено не более Q из этих буферов в скобках, поскольку придется передать $D-1$ прочих буферов в группу активных буферов чтения сразу после того, как будет получен сигнал о готовности к вводу. Любой дополнительный буфер из группы прочих может явно очищаться, как если бы содержащаяся в нем информация еще и не считывалась. Такая очистка выполняется, например, в цикле 4 в (24): невозможно обработать все шесть блоков $d_2 e_2 d_3 f_1 g_1 a_2$ в течение цикла 5, поскольку $Q = 4$. Так что придется повторно прочесть g_1 и a_2. В противном же случае чтение выполняется на полной скорости.

В упр. 29 доказано, что для любой заданной хронологической последовательности серий, которые должны быть слиты, метод рандомизированного разделения позволяет достичь в среднем минимального числа операций чтения с дисков, пропорционального $r(D, Q+2)$, где функция r табулирована в табл. 2. Например, если $D = 4$ и $Q = 18$, среднее время выполнения P-путевого слияния L блоков данных с использованием 4 дисков и $P + 25$ входных буферов будет не больше времени

чтения $r(4,20)L/D \approx 1.785L/4$ блоков с единственного диска. Эта теоретическая оценка верхней границы довольно консервативна; на практике результаты бывают даже лучше — они колеблются в пределах оптимального времени $L/4$.

Таблица 2

ГАРАНТИРОВАННАЯ ПРОИЗВОДИТЕЛЬНОСТЬ ПРИ РАНДОМИЗИРОВАННОМ РАЗДЕЛЕНИИ

	$r(d,d)$	$r(d,2d)$	$r(d,3d)$	$r(d,4d)$	$r(d,5d)$	$r(d,6d)$	$r(d,7d)$	$r(d,8d)$	$r(d,9d)$	$r(d,10d)$
$d=2$	1.500	1.500	1.499	1.467	1.444	1.422	1.393	1.370	1.353	1.339
$d=4$	2.460	2.190	1.986	1.888	1.785	1.724	1.683	1.633	1.597	1.570
$d=8$	3.328	2.698	2.365	2.183	2.056	1.969	1.889	1.836	1.787	1.743
$d=16$	4.087	3.103	2.662	2.434	2.277	2.156	2.067	1.997	1.933	1.890
$d=32$	4.503	3.392	2.917	2.654	2.458	2.319	2.218	2.130	2.062	2.005
$d=64$	5.175	3.718	3.165	2.847	2.613	2.465	2.346	2.249	2.174	2.107
$d=128$	5.431	3.972	3.356	2.992	2.759	2.603	2.459	2.358	2.273	2.201
$d=256$	5.909	4.222	3.536	3.155	2.910	2.714	2.567	2.464	2.363	2.289
$d=512$	6.278	4.455	3.747	3.316	3.024	2.820	2.675	2.556	2.450	2.375
$d=1024$	6.567	4.689	3.879	3.434	3.142	2.937	2.780	2.639	2.536	2.452

Поможет ли сортировка ключей? Если записи длинные, а ключи короткие, весьма заманчиво создать новый файл, состоящий только из ключей с порядковыми номерами, определяющими их первоначальное положение в файле. После сортировки этого файла ключей можно заменить ключи последовательными числами $1, 2, \ldots$. Затем новый файл можно рассортировать по положению в первоначальном файле, в результате чего получится удобное описание того, как "растасовать" записи, т. е. окончательно перекомпоновать их в требуемом порядке. Схематически представим процесс так.

i)	Исходный файл	$(K_1, I_1)(K_2, I_2)\ldots(K_N, I_N)$	Длинный
ii)	Файл ключей	$(K_1, 1)(K_2, 2)\ldots(K_N, N)$	Короткий
iii)	Рассортированный (ii)	$(K_{p_1}, p_1)(K_{p_2}, p_2)\ldots(K_{p_N}, p_N)$	Короткий
iv)	Отредактированный (iii)	$(1, p_1)(2, p_2)\ldots(N, p_N)$	Короткий
v)	Рассортированный (iv)	$(q_1, 1)(q_2, 2)\ldots(q_N, N)$	Короткий
vi)	Отредактированный (i)	$(q_1, I_1)(q_2, I_2)\ldots(q_N, I_N)$	Длинный

Здесь $p_j = k$ тогда и только тогда, когда $q_k = j$. Два процесса сортировки на стадиях (iii) и (v) сравнительно быстрые (иногда можно обойтись внутренней сортировкой), так как записи не слишком длинные. На стадии (vi) задача сведена к сортировке файла, ключи которого — просто числа $\{1, 2, \ldots, N\}$. Каждая запись теперь определяет в точности то место, куда ее следует переместить.

Внешняя перекомпоновка записей, остающаяся после стадии (vi), на первый взгляд кажется тривиальной, но фактически она очень сложна, и все еще не найдено ни одного действительно хорошего алгоритма (существенно лучшего, чем алгоритм для сортировки). Мы, очевидно, могли бы выполнить перекомпоновку за N шагов, перемещая всякий раз по одной записи. Для достаточно большого N это лучше, чем $N \log N$ в методах сортировки, хотя N никогда не бывает таким большим. Но N все-таки достаточно велико, чтобы сделать N поисков просто нереальными.

К отредактированным записям можно эффективно применять какой-либо метод поразрядной сортировки, поскольку ключи имеют строго равномерное распределение. На многих современных быстродействующих компьютерах время вычислений для 8-путевого распределения значительно меньше, чем время вычислений для 8-путевого слияния. Следовательно, распределительная сортировка может быть наилучшей из всех процедур (см. раздел 5.4.7, а также упр. 19).

Но, с другой стороны, представляется уж слишком расточительным выполнять сортировку ключей после того, как они уже были рассортированы. Одна из причин возникновения проблемы, связанной с внешним переразмещением, была открыта Р. У. Флойдом, который нашел нетривиальную нижнюю границу для числа поисков, необходимых для перестановки записей в дисковом файле [*Complexity of Computer Computations* (New York: Plenum, 1972), 105–109].

Результат Флойда удобно описать, воспользовавшись аналогией с лифтом, как в разделе 5.4.8. На этот раз найдем график работы лифта, минимизирующий число *остановок*, а не пройденное расстояние. (Минимизация числа остановок не вполне эквивалентна нахождению алгоритма перегруппировки с минимальным числом поисков, так как одна остановка объединяет вход в лифт с выходом из него. Однако разница не слишком велика, и мы воспользуемся критерием минимизации остановок, чтобы пояснить основные идеи.)

Будем использовать функцию дискретной энтропии

$$F(n) = \sum_{1<k\le n} \bigl(\lceil \lg k \rceil + 1\bigr) = B(n) + n - 1 = n\lceil \lg n \rceil - 2^{\lceil \lg n \rceil} + n, \tag{25}$$

где $B(n)$ есть функция бинарной вставки (см. формулу 5.3.1–(3)). Согласно 5.3.1–(34) функция $F(n)$ — это не что иное, как минимальная длина внешнего пути бинарного дерева с n листьями, и

$$n \lg n \le F(n) \le n \lg n + 0.0861n. \tag{26}$$

Так как $F(n)$ выпукла и удовлетворяет условию $F(n) = n + F\bigl(\lfloor n/2 \rfloor\bigr) + F\bigl(\lceil n/2 \rceil\bigr)$, согласно сформулированной выше лемме C

$$F(n) \le F(k) + F(n-k) + n \qquad \text{при } 0 \le k \le n. \tag{27}$$

Это соотношение вытекает также из представления F в виде длины внешнего пути; в дальнейшем оно окажется решающим аргументом в наших рассуждениях.

Как и в разделе 5.4.8, предположим, что лифт вмещает m человек, каждый этаж вмещает b человек и имеется n этажей. Пусть s_{ij} — число людей, находящихся в данный момент на этаже i и стремящихся попасть на этаж j. По определению *оценка совместности* любого расположения людей в здании есть $\sum_{1\le i,j\le n} F(s_{ij})$.

Предположим, например, что $b = m = n = 6$ и что первоначально 36 человек следующим образом распределены между этажами:

```
␣␣␣␣␣␣
123456  123456  123456  123456  123456  123456.
```
(28)

Лифт пуст и находится на этаже 1; "␣" обозначает свободное место. На каждом этаже имеется по одному человеку, стремящемуся попасть на каждый из этажей,

поэтому все величины s_{ij} равны 1 и оценка совместности равна 0. Если теперь лифт отвезет 6 человек на этаж 2, то мы получим расположение

$$\begin{array}{cccccc} & 123456 & & & & \\ \sqcup\sqcup\sqcup\sqcup\sqcup\sqcup & 123456 & 123456 & 123456 & 123456 & 123456 \end{array} \tag{29}$$

и оценка совместности станет равной $6F(0) + 24F(1) + 6F(2) = 12$. Допустим, лифт перевезет 1, 1, 2, 3, 3 и 4 на этаж 3:

$$\begin{array}{cccccc} & & 112334 & & & \\ \sqcup\sqcup\sqcup\sqcup\sqcup\sqcup & 245566 & 123456 & 123456 & 123456 & 123456 \end{array} \tag{30}$$

Оценка совместности изменится до $4F(2) + 2F(3) = 18$. Когда каждый пассажир будет, в конце концов, перевезен к своему месту назначения, оценка совместности станет равной $6F(6) = 96$.

Флойд заметил, что оценка совместности никогда не увеличивается больше чем на $b + m$ за одну остановку, так как множество из s пассажиров с одним пунктом назначения, объединяясь с аналогичным множеством мощности s', увеличивает оценку на $F(s + s') - F(s) - F(s') \leq s + s'$. Таким образом, имеем следующий результат.

Теорема F. *Пусть t — определенная выше оценка совместности начального расположения пассажиров. Лифт должен сделать по крайней мере*

$$\lceil (F(b)n - t)/(b + m) \rceil$$

остановок, чтобы доставить всех пассажиров к их месту назначения. ∎

Сформулируем этот результат применительно к дискам. Допустим, что имеется bn записей по b в каждом блоке, и предположим, что в оперативной памяти одновременно помещается m записей. При каждом чтении с диска один блок переносится в память, а при каждой записи один блок помещается на диск, и s_{ij} есть число записей в блоке i, принадлежащих блоку j. Если $n \geq b$, то существует начальное расположение, для которого все $s_{ij} \leq 1$, так что $t = 0$. Следовательно, для переразмещения записей необходимо выполнить хотя бы $f(b)n/(b + m) \approx (bn \lg b)/m$ операций чтения блока. (Если b велико, то множитель $\lg b$ делает эту нижнюю оценку нетривиальной.) В упр. 17 выведена несколько более строгая нижняя оценка для общего случая, когда m существенно больше b.

УПРАЖНЕНИЯ

1. [*M22*] В тексте раздела анализируется метод, с помощью которого среднее время ожидания, необходимое для чтения доли x дорожки, уменьшается с $\frac{1}{2}$ до $\frac{1}{2}(1 - x^2)$ оборотов. Это минимально возможное время, когда имеется один держатель головок. Каково соответствующее минимальное время ожидания, если имеется *два* держателя головок, разнесенных на 180° (в предположении, что в любой момент данные могут передаваться только через головки на одном из держателей)?

2. [*M30*] (А. Г. Конхейм (A. G. Konheim).) Назначение этого упражнения — исследовать, на какое расстояние должен переместиться держатель головок во время слияния файлов, расположенных "ортогонально" к цилиндрам. Допустим, имеется P файлов, в каждом из которых содержится L блоков записей, и предположим, что первый блок каждого файла находится на цилиндре 1, второй — на цилиндре 2 и т. д. Движением

держателя головок во время слияния управляет относительный порядок последних ключей каждого блока; следовательно, можно изобразить ситуацию следующим способом, удобным для математической интерпретации. Рассмотрим множество из PL упорядоченных пар

$$\begin{matrix} (a_{11},1) & (a_{21},1) & \dots & (a_{P1},1) \\ (a_{12},2) & (a_{22},2) & \dots & (a_{P2},2) \\ \vdots & \vdots & & \vdots \\ (a_{1L},L) & (a_{2L},L) & \dots & (a_{PL},L) \end{matrix},$$

где множество $\{a_{ij} \mid 1 \le i \le P,\ 1 \le j \le L\}$ состоит из чисел $\{1,2,\dots,PL\}$ в некотором порядке и $a_{ij} < a_{i(j+1)}$ при $1 \le j < L$ (строки представляют цилиндры, столбцы — исходные файлы). Рассортируем пары по их первым компонентам и будем считать, что получилась последовательность $(1,j_1)\,(2,j_2)\dots(PL,j_{PL})$. Покажите, что если все $(PL)!/L!^P$ возможных наборов a_{ij} равновероятны, то среднее значение

$$|j_2-j_1|+|j_3-j_2|+\cdots+|j_{PL}-j_{PL-1}|$$

равно

$$(L-1)\left(1+(P-1)2^{2L-2}\Big/\binom{2L}{L}\right).$$

[*Указание.* См. упр. 5.2.1–14.] Заметим, что при $L \to \infty$ это значение асимптотически стремится к $\frac{1}{4}(P-1)L\sqrt{\pi L}+O(PL)$.

3. [*M15*] Предположим, что ограниченный размер внутренней памяти делает нежелательным 10-путевое слияние. Как можно изменить рекуррентные соотношения (3)–(5), чтобы $A_1(n)$ имело минимальное значение $\alpha D(\mathcal{T})+\beta E(\mathcal{T})$ среди всех деревьев $\mathcal{T}$ с n листьями, не имеющих узлов степени, большей, чем 9?

▶ **4.** [*M21*] Рассмотрим модифицированный вариант квадратичной схемы выделения буферов, при которой все P входных буферов имеют равные объемы, а объем выходного буфера нужно выбрать из соображения минимизации времени поиска.

a) Выведите формулу, аналогичную (2), для времени выполнения P-путевого слияния L символов.

b) Покажите, что построение в теореме K можно изменить так, что получится схема слияния, которая будет оптимальной в смысле вашей формулы из п. (a).

5. [*M20*] Если используются два диска, причем чтение с одного совмещается с записью на другой, то нельзя применять схему слияния, подобную представленной на рис. 93, так как одни листья находятся на четных уровнях, а другие — на нечетных. Покажите, как изменить построение в теореме K, чтобы получались оптимальные деревья с учетом ограничения, что все листья находятся на четных уровнях или все на нечетных.

▶ **6.** [*22*] Найдите дерево, оптимальное в смысле упр. 5, если $n = 23$ и $\alpha = \beta = 1$. (Возможно, у вас появится желание прибегнуть к помощи компьютера.)

▶ **7.** [*M24*] Если длины начальных серий не равны, то наилучшая схема слияния (в смысле теоремы H) минимизирует $\alpha D(\mathcal{T})+\beta E(\mathcal{T})$, где $D(\mathcal{T})$ и $E(\mathcal{T})$ теперь представляют собой *взвешенные* длины путей: каждому листу дерева приписываются веса $w_1,\dots,w_n$ (соответствующие длинам начальных серий), а суммы степеней и длины путей умножаются на соответствующие веса. Например, если $\mathcal{T}$ — дерево, представленное на рис. 92, то получим $D(\mathcal{T}) = 6w_1+6w_2+7w_3+9w_4+9w_5+7w_3+4w_7+4w_8$, $E(\mathcal{T}) = 2w_1+2w_2+2w_3+3w_4+3w_5+2w_6+w_7+w_8$.

Докажите, что при некотором k всегда существует оптимальная схема, в которой первыми сливаются k *самых коротких* серий.

8. [*49*] Существует ли алгоритм, который при заданных α, β и весах $w_1, \ldots, w_n$ находит оптимальные в смысле упр. 7 деревья, затрачивая только $O(n^c)$ шагов при некотором c?

9. [*HM39*] (Л. Хьяфил (L. Hyafil), Ф. Прускер (F. Prusker) и Ж. Вуйлемен (J. Vuillemin).) Докажите, что для фиксированных α и β получается

$$A_1(n) = \left(\min_{m\geq 2} \frac{\alpha m + \beta}{\log m}\right) n \log n + O(n)$$

при $n \to \infty$, где член $O(n) \geq 0$.

10. [*HM44*] (Л. Хьяфил, Ф. Прускер и Ж. Вуйлемен.) Докажите, что при фиксированных α и β $A_1(n) = \alpha mn + \beta n + A_m(n)$ для всех достаточно больших n, если m минимизирует коэффициент в упр. 9.

11. [*M29*] В обозначениях (6) и (11) докажите, что $f_m(n) + mn \geq f(n)$ при всех $m \geq 2$ и $n \geq 2$, и определите все m и n, при которых имеет место равенство.

12. [*25*] Докажите, что при всех $n > 0$ существует дерево с n листьями и минимальной длиной степенного пути соответственно (6), все листья которого расположены на одном уровне.

13. [*M24*] Покажите, что при $2 \leq n \leq d(\alpha, \beta)$, где $d(\alpha, \beta)$ определено в соотношении (12), единственной наилучшей схемой слияния в смысле теоремы H является n-путевое слияние.

14. [*40*] При использовании квадратичного метода распределения буферов время поиска для схемы слияния, представленной на рис. 92, будет пропорционально $(\sqrt{2}+\sqrt{4}+\sqrt{1}+\sqrt{1}+\sqrt{8})^2 + (\sqrt{1}+\sqrt{1}+\sqrt{2})^2 + (\sqrt{1}+\sqrt{2}+\sqrt{1}+\sqrt{4})^2 + (\sqrt{1}+\sqrt{1}+\sqrt{2})^2$. Это значение представляет собой сумму величин $(\sqrt{n_1}+\cdots+\sqrt{n_m}+\sqrt{n_1+\cdots+n_m})^2$ по всем внутренним узлам, где поддеревья данных узлов имеют $(n_1, \ldots, n_m)$ листьев. Напишите программу, которая порождает деревья с минимальным временем поиска, имеющие 1, 2, 3, ... листьев, используя для оценки времени поиска эту формулу.

15. [*M22*] Покажите, что можно несколько "усилить" теорему F, если лифт первоначально пуст и если $F(b)n \neq t$. В этом случае необходимо не менее $\lceil (F(b)n + m - t)/(b+m) \rceil$ остановок.

16. [*23*] (Р. У. Флойд.) Найдите график работы лифта, перевозящего всех людей из (28) к их местам назначения через 12 или менее остановок. (В (29) показано положение после одной, а не после двух остановок.)

▶ **17.** [*HM25*] (Р. Флойд, 1980.) Покажите, что нижняя оценка в теореме F может быть улучшена до

$$\frac{n(b\ln n - \ln b - 1)}{\ln n + b(1 + \ln(1 + m/b))}$$

в том смысле, что некоторая исходная конфигурация должна потребовать, по крайней мере, столько остановок. [*Указание.* Пересчитайте конфигурации, которые могут образоваться после s остановок.]

18. [*HM26*] Пусть L — нижняя оценка из упр. 17. Покажите, что среднее число остановок лифта, необходимое для того, чтобы доставить всех пассажиров по назначению, будет не менее $L - 1$, когда равновероятны все $(bn)!$ возможных перестановок bn пассажиров.

▶ **19.** [*25*] (Б. Т. Беннет (B. T. Bennett) и А. Ч. Мак-Келлар (A. C. McKellar), 1971.) Рассмотрим следующий подход к сортировке ключей, продемонстрированный на примере файла с 10 ключами.

i) Исходный файл: $(50, I_0)(08, I_1)(51, I_2)(06, I_3)(90, I_4)(17, I_5)(89, I_6)(27, I_7)(65, I_8)(42, I_9)$
ii) Файл ключей: $(50, 0)(08, 1)(51, 2)(06, 3)(90, 4)(17, 5)(89, 6)(27, 7)(65, 8)(42, 9)$

iii) Рассортированный файл ключей (ii): $(06,3)(08,1)(17,5)(27,7)(42,9)(50,0)(51,2)(65,8)(89,6)(90,4)$

iv) Присвоение номеров ящиков (см. ниже): $(2,1)(2,3)(2,5)(2,7)(2,8)(2,9)(1,0)(1,2)(1,4)(1,6)$

v) Рассортированные номера ящиков (iv): $(1,0)(2,1)(1,2)(2,3)(1,4)(2,5)(1,6)(2,7)(2,8)(2,9)$

vi) (i) Исходный файл, распределенный по ящикам с использованием рассортированных номеров ящиков (v):
Ящик 1: $(50,I_0)(51,I_2)(90,I_4)(89,I_6)$
Ящик 2: $(08,I_1)(06,I_3)(17,I_5)(27,I_7)(65,I_8)(42,I_9)$

vii) Результат выбора с замещением (сначала читается ящик 2, затем — ящик 1):
$(06,I_3)(08,I_1)(17,I_5)(27,I_7)(42,I_9)(50,I_0)(51,I_2)(65,I_8)(89,I_6)(90,I_4)$

Номера ящиков на шаге (iv) присваиваются путем применения к (iii) *выбора с замещением справа налево в порядке убывания* по второй компоненте. Номер ящика — это номер серии. В приведенном примере используется выбор с замещением только с двумя элементами в дереве. Для выбора с замещением в (iv) и (vii) должен использоваться один и тот же размер дерева выбора. Заметим, что содержимое ящиков необязательно упорядочено.

Докажите, что этот метод позволит выполнять сортировку, т. е. что выбор с замещением в (vii) образует лишь одну серию. (Этот метод уменьшает число ящиков, необходимых для обычной сортировки ключей посредством распределения, особенно если исходные данные уже в значительной степени упорядочены.)

▶ **20.** [*25*] Некоторые системы предоставляют в распоряжение программиста *виртуальную память*: можно писать программы, исходя из предположения, что имеется очень большой объем оперативной памяти, способной вместить все данные. Эта память разделена на страницы, и лишь немногие из них действительно в произвольный момент времени находятся в оперативной памяти; остальные же хранятся на дисках или барабанах. Программисту не нужно вникать во все эти подробности, так как система сама обо всем заботится: новые страницы автоматически вызываются в память, когда в них возникает необходимость.

Может показаться, что появление виртуальной памяти приводит к отмиранию методов внешней сортировки, так как эта работа может быть выполнена с помощью методов, предназначенных для внутренней сортировки. Проанализируйте такую ситуацию. Почему специально разработанный метод внешней сортировки может оказаться лучше применения обычной технологии подкачки страниц к методу внутренней сортировки?

▶ **21.** [*M15*] Сколько блоков переходит на диск j из L-блочного файла при разделении на D дисков?

22. [*22*] Пусть сливаются два файла по принципу Гилбрета и желательно сохранить ключи α_j с a блоками и ключи β_j с b блоками. В какой блок необходимо поместить α_j для того, чтобы получить при необходимости нужную информацию?

▶ **23.** [*20*] Какой объем памяти необходим для входных буферов, в которых предполагается довольно продолжительно хранить исходные данные, если выполняется 2-путевое слияние посредством (a) разделения суперблоков; (b) принципа Гилбрета?

24. [*M36*] Предположим, что P серий разделены между D дисками так, что блок j серии k находится на диске $(x_k + j) \bmod D$. При выполнении P-путевого слияния эти блоки будут считываться в некотором хронологическом порядке, подобном (19). Если группы из D блоков должны вводится продолжительное время, то в момент времени t мы будем считывать с каждого диска t-й в хронологическом порядке блок, как в (21). Каков минимальный объем буфера в памяти (в количестве записей), необходимый для хранения исходных данных, которые еще не были вовлечены в процесс слияния, если не учитывать хронологический

порядок? Объясните, как выбрать смещения $x_1, x_2, \ldots, x_P$ с тем, чтобы в худшем случае требовалось как можно меньше буферов.

25. [*23*] Повторите пример из текста раздела, рассмотренный в связи с рандомизированным разделением, изменив условия: пусть $Q = 3$ вместо $Q = 4$. Что окажется в буферах в отличие от (24)?

26. [*26*] Сколько выходных буферов необходимо, чтобы выполнение P-путевого слияния с рандомизированным разделением никогда не было приостановлено из-за отсутствия места в памяти для размещения только что полученных в результате слияния данных? Считайте, что время записи блока равно времени считывания.

27. [*HM27*] (*Проблема циклической занятости.*) Предположим, что n пустых урн расставлены по кругу и им присвоены номера $0, 1, \ldots, n-1$. При $k = 1, 2, \ldots, p$ мы бросаем m_k шаров в урны $(X_k + j) \bmod n$, $j = 0, 1, \ldots, m_k - 1$, где целые числа X_k выбираются случайно. Пусть $S_n(m_1, \ldots, m_p)$ — число шаров в урне 0, а $E_n(m_1, \ldots, m_p)$ — ожидаемое число шаров в самой заполненной урне.

a) Докажите, что $E_n(m_1, \ldots, m_p) \le \sum_{t=1}^{m} \min(1,\ n \Pr(S_n(m_1, \ldots, m_p) \ge t))$, где $m = m_1 + \cdots + m_p$.

b) Используя неравенство в соотношении 1.2.10–(25), докажите, что

$$E_n(m_1, \ldots, m_p) \le \sum_{t=1}^{m} \min\Bigl(1,\ \frac{n(1+\alpha_t/n)^m}{(1+\alpha_t)^t}\Bigr)$$

для любых неотрицательных действительных чисел $\alpha_1, \alpha_2, \ldots, \alpha_m$. Какие значения $\alpha_1, \ldots, \alpha_m$ дадут наилучшую верхнюю оценку?

28. [*HM47*] Продолжая упр. 27, ответьте, справедливо ли неравенство $E_n(m_1, \ldots, m_p) \ge E_n(m_1 + m_2, m_3, \ldots, m_p)$?

▶ **29.** [*M30*] Назначение этого упражнения — вывести верхнюю оценку среднего времени, необходимого для ввода любой последовательности блоков в хронологическом порядке при выполнении процедуры рандомизированного разделения, когда блоки представляют P серий и D дисков. Мы говорим, что блок, ожидаемый на некотором временном цикле в то время, когда выполняется алгоритм (см. (24)), является "маркированным"; таким образом, суммарное время ввода пропорционально числу маркированных блоков. Маркирование зависит только от хронологической последовательности доступа к блокам (см. (20)).

a) Докажите, что если $Q+1$ последовательных блоков в хронологическом порядке имеют N_j блоков на диске j, то не более $\max(N_0, N_1, \ldots, N_{D-1})$ из них являются маркированными.

b) Усильте результат (a), показав, что он имеет место также для $Q+2$ последовательных блоков.

c) Теперь используйте результаты анализа проблемы циклической занятости из упр. 27 и получите верхнюю оценку среднего времени выполнения в терминах функции $r(D, Q+2)$, как в табл. 2, задав любой хронологический порядок.

30. [*HM30*] Докажите, что для функции $r(d, m)$ из упр. 29 при $s \to \infty$ выполняется условие $r(d, sd \log d) = 1 + O(1/\sqrt{s})$ для фиксированного d.

31. [*HM48*] Проанализируйте рандомизированное разделение и определите поведение этого алгоритма в среднем, а не только по отношению к верхней оценке, как функцию от P, Q и D. (Интересен даже случай для $Q = 0$, когда требуется в среднем $\Theta(L/\sqrt{D})$ циклов чтения.)

5.5. РЕЗЮМЕ. ИСТОРИЯ И БИБЛИОГРАФИЯ

Теперь, когда мы почти достигли конца этой очень длинной главы, "рассортируем" наиболее важные из рассмотренных выше сведений.

Алгоритм сортировки — это процедура, которая реорганизует файл записей таким образом, что ключи оказываются расположенными в порядке возрастания. Вследствие такого упорядочения группируются записи с равными ключами, становится возможной эффективная обработка множества файлов, рассортированных по одному ключу, создается информационная основа для эффективных алгоритмов выборки и более убедительно выглядят документы, подготовленные с помощью компьютера.

Внутренняя сортировка используется в тех случаях, когда все записи помещаются в быстродействующей внутренней памяти компьютера. Мы изучили с разной степенью детализации более двух десятков алгоритмов внутренней сортировки. Но, наверное, было бы куда спокойнее, если бы мы не знали столько различных подходов к решению данной задачи! Изучение всех этих методов было приятным времяпрепровождением. Но теперь перед нами безрадостная перспектива — предстоит на деле решить, какой метод следует использовать в той или иной конкретной ситуации.

Было бы прекрасно, если бы один или два метода сортировки превосходили все остальные безотносительно к приложению или используемой машине. На самом же деле каждый метод имеет свои собственные, одному ему присущие достоинства. Например, метод пузырька (алгоритм 5.2.2B) не имеет ярко выраженных преимуществ, так как всегда можно найти лучший способ сделать то, что он делает; но даже он после соответствующего обобщения оказывается полезным для сортировки с двумя лентами (см. раздел 5.4.8). Итак, приходим к заключению, что почти все алгоритмы заслуживают того, чтобы о них помнили, так как существуют приложения, в которых какой-либо из них оказывается самым подходящим.

В следующем кратком обзоре освещаются основные аспекты наиболее важных алгоритмов внутренней сортировки, с которыми мы встречались. (Как обычно, N означает число записей в файле.)

1. *Распределяющий подсчет.* Если диапазон ключей невелик, очень полезен алгоритм 5.2D. Метод устойчив (не изменяет порядок записей с равными ключами), но требуется память для счетчиков и $2N$ записей. Одна из модификаций, позволяющая сэкономить N из этих записей ценой устойчивости, встречается в упр. 5.2–13.

2. *Простая вставка.* Алгоритм 5.2.1S наиболее прост для программной реализации, не требует дополнительного объема памяти и довольно эффективен при малых N (скажем, при $N \leq 25$). При больших N он становится невыносимо медленным, если только исходные данные не окажутся сразу почти упорядоченными.

3. *Сортировка с убывающим смещением.* Алгоритм 5.2.1D (метод Шелла) также довольно просто программируется, использует минимальный объем памяти и весьма эффективен при умеренно больших N (скажем, при $N \leq 1000$).

4. *Вставка в список.* Алгоритм 5.2.1L основан на той же идее, что и алгоритм простой вставки, и поэтому годится только при небольших N. Как и в других методах сортировки списков, в нем благодаря операциям со ссылками экономятся затраты на пересылку длинных записей; это особенно удобно, когда записи имеют переменную длину или являются частью других структур данных.

5. *Сортировка с вычислением адреса* эффективна, если ключи подчиняются известному (обычно — равномерному) закону распределения; важнейшими вариантами этого подхода являются *вставки в несколько списков* (программа 5.2.1M) и комбинированная поразрядная сортировка со вставками Мак-Ларена (рассмотренная в конце раздела 5.2.5). Для последнего метода достаточно иметь $O(\sqrt{N})$ дополнительных ячеек памяти. Двухпроходный метод, который позволяет иметь дело с неравномерным распределением, анализируется в теореме 5.2.5T.

6. *Обменная сортировка со слиянием.* Алгоритм 5.2.2M (метод Бэтчера) и родственный ему алгоритм *битонной сортировки* (упр. 5.3.4–10) полезны, если можно одновременно выполнять большое число сравнений.

7. *Обменная сортировка с разделением* (метод Хоара, широко известный как быстрая сортировка). Алгоритм 5.2.2Q, вероятно, — самый полезный универсальный алгоритм внутренней сортировки, поскольку он требует очень мало памяти и опережает своих конкурентов по среднему времени выполнения на большинстве компьютеров. Однако в наихудшем случае он может работать *очень* медленно. Поэтому, если вероятность неслучайных данных достаточно велика, приходится тщательно выбирать разделяющие элементы. Если выбирается медиана из трех элементов (как предлагается в упр. 5.2.2–55), то такое поведение, как в наихудшем случае, становится крайне маловероятным и, кроме того, несколько уменьшается среднее время работы.

8. *Простой выбор.* Алгоритм 5.2.3S довольно прост и особенно подходит в случае, когда имеется специальное оборудование для быстрого поиска наименьшего элемента в списке.

9. *Пирамидальная сортировка.* Алгоритм 5.2.3H при минимальных требованиях к памяти обеспечивает достаточно высокую скорость сортировки; как среднее, так и максимальное время работы примерно вдвое больше среднего времени быстрой сортировки.

10. *Слияние списков.* Алгоритм 5.2.4L осуществляет сортировку списков, обеспечивая при этом, так же как и алгоритм пирамидальной сортировки, весьма высокую скорость даже в наихудшем случае. Кроме того, данный метод устойчив по отношению к равным ключам.

11. *Поразрядная сортировка* с использованием алгоритма 5.2.5R — это не что иное, как сортировка списков, которая приемлема для ключей либо очень коротких, либо имеющих необычный порядок лексикографического сравнения. Вместо ссылок можно применить распределяющий подсчет (п. 1 нашего обзора); такая процедура потребует пространства для $2N$ записей и для таблицы счетчиков, но благодаря простоте внутреннего цикла она особенно хороша для сверхбыстрых компьютеров — "пожирателей чисел", имеющих опережающее управление. (*Предостережение.* Поразрядную сортировку не следует использовать при малых N!)

12. *Сортировка методом вставок и слияния* (см. раздел 5.3.1) наиболее приемлема при очень малых N в "прямолинейных" программах. Например, этот метод оказался бы подходящим в тех приложениях, в которых требуется сортировать много групп из пяти или шести записей.

13. Могут существовать и гибридные методы, объединяющие один или более из приведенных выше. Например, короткие подфайлы, возникающие при быстрой сортировке, можно сортировать методом слияния и вставок.

14. И наконец, для реализации безымянного метода, встретившегося в ответе к упр. 5.2.1–3, требуется, по-видимому, кратчайшая из возможных программ сортировки. Но среднее время работы такой программы пропорционально N^3, т. е. это самая медленная программа сортировки из упомянутых в данной книге!

Пространственные и временные характеристики многих из этих методов, запрограммированных для компьютера MIX, сведены в табл. 1. Важно иметь в виду, что числа в данной таблице являются лишь грубыми оценками относительного времени сортировки. Они применимы только к одному компьютеру, и предположения, касающиеся исходных данных, далеко не для всех программ абсолютно правомерны. Сравнительные таблицы, подобные этой, приводились во многих работах, но не найдется таких двух авторов, которые пришли бы к одинаковым выводам! Тем не менее данные о времени работы позволяют оценить хотя бы порядок скорости, которую следует ожидать от каждого алгоритма при сортировке записей из одного слова, так как MIX — довольно типичный компьютер.

В столбце "Память" в табл. 1 содержится некоторая информация об объеме вспомогательной памяти, используемой каждым алгоритмом, в единицах длины записи. Здесь буквой ϵ обозначена доля записи, необходимая для одного поля связи; так, например, $N(1+\epsilon)$ означает, что методу требуется пространство для N записей и N полей связи.

В асимптотических оценках среднего и максимального времени, приведенных в табл. 1, учитываются только главные члены, доминирующие при больших N в предположении случайных исходных данных; c обозначает произвольную константу. Эти формулы могут иногда ввести в заблуждение, поэтому указано также фактическое время выполнения программы для двух конкретных последовательностей исходных данных. Случай $N = 16$ относится к шестнадцати ключам, так часто появлявшимся в примерах раздела 5.2, а случай $N = 1000$ относится к последовательности $K_1, K_2, \ldots, K_{1000}$, определенной формулами

$$K_{1001} = 0; \qquad K_{n-1} = (3141592621K_n + 2113148651) \bmod 10^{10}.$$

Для получения характеристик каждого алгоритма, представленного в таблице, использовалась достаточно совершенная программа для MIX, как правило, учитывающая усовершенствования, которые описаны в упражнениях. Размер байта при выполнении этих программ принят равным 100.

Для внешней сортировки необходимы методы, отличающиеся от методов внутренней сортировки, потому что предполагается использование сравнительно простых структур данных и большое внимание уделяется уменьшению времени ввода-вывода. В разделе 5.4.6 рассматриваются интересные методы, разработанные для сортировки данных на магнитных лентах, а в разделе 5.4.9 обсуждается использование дисков и барабанов.

Конечно, сортировка — не единственная тема этой главы. Попутно мы много узнали о том, как работать со структурами данных, обращаться с внешней памятью, анализировать алгоритмы, и о том, как изобретать... новые алгоритмы.

Таблица 1

СРАВНЕНИЕ МЕТОДОВ ВНУТРЕННЕЙ СОРТИРОВКИ ПРИМЕНИТЕЛЬНО К РЕАЛИЗАЦИИ НА КОМПЬЮТЕРЕ MIX

Метод	Где упоминается	Устойчив?	Длина кода MIX	Память	Время выполнения				Прим.
					Среднее	Макс.	$N = 16$	$N = 1000$	
Сравнение и подсчет	Упр. 5.2–5	Да	22	$N(1+\epsilon)$	$4N^2 + 10N$	$5.5N^2$	1065	3992432	c
Распределение и подсчет	Упр. 5.2–9	Да	26	$2N + 1000\epsilon$	$22N + 10010$	$22N$	10362	32010	a
Простая вставка	Упр. 5.2.1–33	Да	10	$N + 1$	$1.5N^2 + 9.5N$	$3N^2$	412	1491928	
Сортировка с убывающим смещением	Прог. 5.2.1D	Нет	21	$N + \epsilon \lg N$	$3.9N^{7/6} + 10N \lg N + 166N$	$cN^{4/3}$	567	128758	d, h
Вставка в список	Упр. 5.2.1–33	Да	19	$N(1+\epsilon)$	$1.25N^2 + 13.25N$	$2.5N^2$	433	1248615	b, c
Вставка в несколько списков	Прог. 5.2.1M	Нет	18	$N + \epsilon(N + 100)$	$.0175N^2 + 18N$	$3.5N^2$	645	35246	b, c, f, i
Обменная сортировка со слиянием	Упр. 5.2.2–12	Нет	35	N	$2.875N(\lg N)^2$	$4N(\lg N)^2$	939	284366	
Обменная сортировка с разделением	Прог. 5.2.2Q	Нет	63	$N + 2\epsilon \lg N$	$11.67N \ln N - 1.74N$	$\geq 2N^2$	470	81486	
Сортировка методом медианы из трех ключей	Упр. 5.2.2–55	Нет	100	$N + 2\epsilon \lg N$	$10.63N \ln N + 2.12N$	$\geq N^2$	487	74574	e
Обменная поразрядная сортировка	Прог. 5.2.2R	Нет	45	$N + 68\epsilon$	$14.43N \ln N + 23.9N$	$272N$	1135	137614	g, i, j
Простой выбор	Прог. 5.2.3S	Нет	15	N	$2.5N^2 + 3N \ln N$	$3.25N^2$	853	2525287	j
Пирамидальная сортировка	Прог. 5.2.3H	Нет	30	N	$23.08N \ln N + 0.01N$	$24.5N \ln N$	1068	159714	h, j
Слияние списков	Прог. 5.2.4L	Да	44	$N(1+\epsilon)$	$14.43N \ln N + 4.92N$	$14.4N \ln N$	761	104716	b, c, j
Поразрядная сортировка списков	Прог. 5.2.5R	Да	36	$N + \epsilon(N + 200)$	$32N + 4838$	$32N$	4250	36838	b, c

Примечания к табл. 1

a Ключи только из трех цифр

b Ключи только из шести цифр (т. е. из трех байтов)

c Вывод не перекомпонован; результирующая последовательность определяется неявно ссылками или счетчиками

d Смещение выбирается, как в 5.2.1–(11); несколько лучшая последовательность появляется в упр. 5.2.1–29

e $M = 9$, если использовать SRB; для варианта с DIV добавить к среднему времени выполнения $1.60N$

f $M = 100$ (размер байта)

g $M = 34$, так как $2^{34} > 10^{10} > 2^{33}$

h Поскольку теория неполная, данные о среднем времени получены эмпирически

i Оценка среднего времени базируется на предположении о равномерном распределении ключей

j Дальнейшее совершенствование программы, о котором упоминается в тексте раздела и упражнениях, относящихся к этой программе, могут уменьшить время выполнения

Ранние разработки. Поиск прототипов современных методов сортировки возвращает нас в 19 век, когда были изобретены первые машины для сортировки. В Соединенных Штатах Америки перепись всех граждан проводилась каждые 10 лет, и уже к 1880 году проблема, связанная с обработкой огромных по объему данных переписи стала очень острой. В самом деле, число одиноких (т. е. не состоящих в браке) граждан не подсчитывалось ежегодно, хотя вся необходимая информация собиралась. Герман Холлерит (Herman Hollerith), двадцатилетний служащий Бюро переписи, изобрел остроумный электрический табулятор, отвечающий нуждам сбора статистики, и около ста его машин успешно использовались при обработке данных переписи 1890 года.

На рис. 94 изображен первый аппарат Холлерита, приводимый в действие от аккумуляторных батарей. Для нас основной интерес представляет "сортировальный ящик" справа, который открыт для того, чтобы можно было увидеть половину из 26 внутренних отделений. Оператор вставлял перфокарту размером $6\frac{5}{8} \times 3\frac{1}{4}$ дюймов в "пресс" и опускал рукоятку; это приводило к тому, что закрепленные на пружинах штыри на верхней панели в тех местах, где на карте пробиты отверстия, входят в контакт с ртутью на нижней панели. В результате замыкания соответствующей электрической цепи показание связанного с ней циферблата изменялось на 1 и, кроме того, одна из 26 крышек сортировального ящика открывалась. В этот момент оператор отпускал пресс, клал карту в открытое отделение и закрывал крышку. Однажды через эту машину пропустили 19 071 карту за один $6\frac{1}{2}$-часовой рабочий день (в среднем около 49 карт в минуту!). (Средний оператор работал примерно втрое медленней.)

Население продолжало неуклонно расти, и первые табуляторы-сортировщики оказались недостаточно быстрыми, чтобы справиться с обработкой данных переписи 1900 года. Поэтому Холлерит изобрел еще одну машину, чтобы предотвратить еще один кризис в обработке данных. Его новое устройство (запатентованное в 1901 и 1904 годах) автоматически подавало карты и выглядело, в сущности, почти так

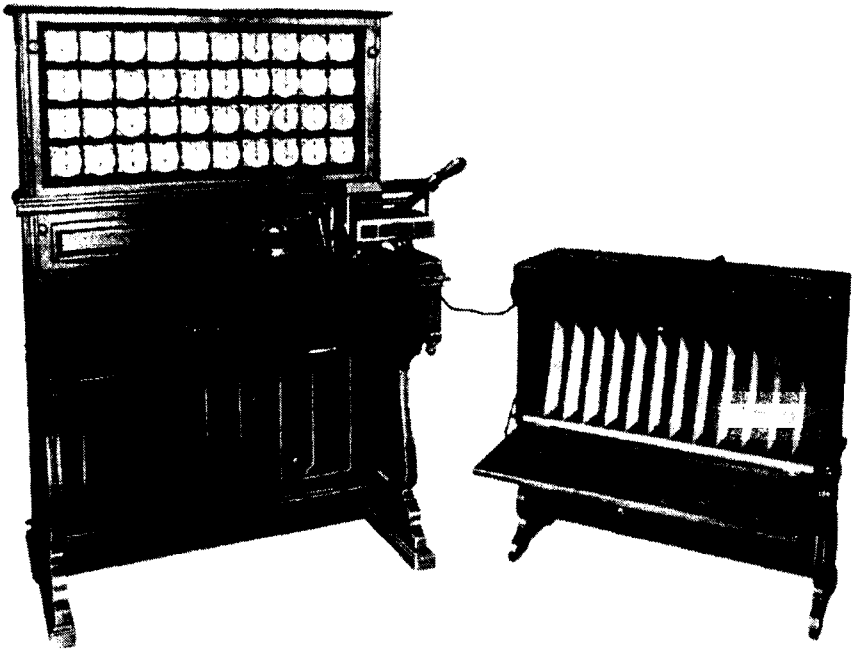


Рис. 94. Табулятор Холлерита. (Фотография любезно предоставлена архивом IBM.)

же, как современные карточные сортировальные машины. История ранних машин Холлерита с интересными подробностями изложена Леоном Э. Трусделлом (Leon E. Truesdell) в книге *The Development of Punch Card Tabulation* (Washington: U. S. Bureau of the Census, 1965); см. также сообщения современников Холлерита: *Columbia College School of Mines Quarterly* **10** (1889), 238–255; *J. Franklin Inst.* **129** (1890), 300–306; *The Electrical Engineer* **12** (November, 11, 1891), 521–530; *J. Amer. Statistical Assn.* **2** (1891), 330–341, **4** (1895), 365; *J. Royal Statistical Soc.* **55** (1892), 326–327; *Allgemeines statistisches Archiv* **2** (1892), 78–126; *J. Soc. Statistique de Paris* **33** (1892), 87–96; *U. S. Patents 395781* (1889), *685608* (1901), *777209* (1904). Холлерит и другой бывший служащий Бюро переписи Джеймс Пауэрс (James Powers) в дальнейшем основали конкурирующие компании, которые, в конце концов, вошли соответственно в корпорации IBM и Remington Rand corporation.

Сортировальная машина Холлерита базировалась, конечно, на методах поразрядной сортировки, используемых в цифровых ЭВМ. В его патенте упоминается, что числовые элементы, содержащие два столбца, должны сортироваться "по отдельности для каждого столбца", но он не говорит, какой столбец (единиц или десятков) должен рассматриваться первым. Патент за номером 518240, выданный Джону К. Гору (John K. Gore) в 1894 году, в котором описана другая машина для сортировки, предполагает начинать со столбца десятков. Далеко не очевидная идея сортировки сначала по столбцу единиц была, по-видимому, открыта каким-то неизвестным оператором и передана остальным (см. раздел 5.2.5); она имеется в самом раннем сохранившемся руководстве IBM по сортировке (1936 г.). Насколько мне известно, впервые метод "справа налево" упоминается в книге Robert Feindler,

Das Hollerith-Lochkarten-Verfahren (Berlin: Reimar Hobbing, 1929), 126–130; почти в это же время он упоминается в статье Л. Дж. Комри (L. J. Comrie), *Transactions of the Office Machinery Users' Association* (London, 1930), 25–37. Кстати, Комри оказался первым, кто сделал следующее важное наблюдение: табуляторы можно с успехом использовать для выполнения научных расчетов, хотя первоначально они создавались для статистических и бухгалтерских приложений. Его статья особенно интересна, поскольку содержит подробное описание табуляторов, имевшихся в Англии в 1930 году. Сортировальные машины в то время обрабатывали от 360 до 400 карт в минуту и сдавались в аренду за 9 фунтов стерлингов в месяц.

Идея слияния восходит к другому устройству для обработки карт — *подборочной машине* (collator), которая была изобретена значительно позднее, в 1938 году. Снабженная двумя подающими механизмами, она могла слить две рассортированные колоды карт в одну всего за один проход; метод выполнения этого слияния хорошо описан в первом руководстве IBM по методам подборки (апрель 1939 года). [См. James W. Bryce, *U. S. Patent 2189024* (1940).]

Затем на сцене появились ЭВМ и разработка методов сортировки тесно переплелась с их развитием. На самом деле имеются свидетельства того, что программа сортировки была первой из когда-либо написанных для вычислительных машин с запоминаемой программой. Конструкторы вычислительной машины EDVAC особенно интересовались сортировкой, поскольку она выступала как наиболее характерный представитель потенциальных нечисловых приложений ЭВМ. Они понимали, что удовлетворительная система команд должна годиться не только для составления программы решения разностных уравнений; она должна обладать достаточной гибкостью, чтобы справиться с комбинаторными аспектами в алгоритмах "выбора решений". Поэтому Джон фон Нейман (John von Neumann) подготовил в 1945 году программы для внутренней сортировки методом слияния, чтобы убедиться в необходимости некоторых типов команд, предложенных им для машины EDVAC. Существование эффективных специализированных сортировальных машин послужило тем естественным стандартом, в сопоставлении с которым можно было оценить достоинства предлагаемой организации электронной вычислительной машины. Подробно это интересное исследование описано в статье D. E. Knuth, *Computing Surveys* **2** (1970), 247–260; первая программа сортировки фон Неймана в окончательном, "отполированном" виде приводится в его сборнике *Collected Works* **5** (New York: Macmillan, 1963), 196–214.

Независимо от них в 1945 году К. Цузе (K. Zuse) в Германии разработал программу для сортировки методом простой вставки. Это был один из самых простых примеров операций с линейными списками в разработанном им языке "Plankalkül". (Эта пионерская работа ожидала публикации почти 30 лет; см. *Berichte der Gesellschaft für Mathematik und Datenverarbeitung* **63** (Bonn, 1972), part 4, 84–85.)

Из-за ограниченного объема памяти в ранних машинах приходилось думать о внешней сортировке наравне с внутренней, и в докладе "Progress Report on the EDVAC", подготовленном Дж. П. Эккертом (J. P. Eckert) и Дж. У. Мочли (J. W. Mauchly) для Школы Мура по электротехнике [Moore School of Electrical Engineering (30 September, 1945)], указывалось, что ЭВМ, оснащенная устройством с магнитной проволокой или лентой, могла бы моделировать действия перфокарточного оборудования, достигая при этом большей скорости сортировки. В этом докладе были

описаны методы сбалансированной двухпутевой поразрядной сортировки и сбалансированного двухпутевого слияния (там он был назван "подборкой" (collating)) с использованием четырех устройств внешней памяти на магнитной проволоке или ленте, читающих или записывающих "не менее 5 000 импульсов в секунду".

Джон Мочли выступил с лекцией о "сортировке и слиянии" на специальной сессии по вычислениям, созывавшейся в Школе Мура в 1946 году. В конспекте его лекции содержится первое опубликованное обсуждение сортировки с помощью вычислительных машин [*Theory and Techniques for the Design of Electronic Digital Computers*, edited by G. W. Patterson, **3** (1946), 22.1–22.20]. Мочли начал свое выступление с интересного замечания: "Требование, чтобы одна машина объединяла возможности вычислений и сортировки, кое-кому может показаться требованием, чтобы один прибор использовался как ключ для консервов и как авторучка". Затем он заметил, что машины, способные выполнять сложные математические процедуры, должны также иметь возможность сортировать и классифицировать данные; он показал, что сортировка может быть полезна даже в связи с численными расчетами. Мочли описал методы простой вставки и бинарных вставок, упомянув, что в первом методе в среднем необходимо около $N^2/4$ сравнений, в то время как в последнем их никогда не требуется более $N \lg N$. Однако для бинарных вставок нужна весьма сложная структура данных и Мочли затем показал, что при двухпутевом слиянии достигается столь же малое число сравнений, но используется только последовательный просмотр списков. Последняя часть конспекта его лекции посвящена методам поразрядной сортировки с частичными проходами, которые моделируют цифровую карточную сортировку на четырех лентах, затрачивая менее четырех проходов на цифру (см. раздел 5.4.7).

Вскоре после этого Эккерт и Мочли организовали компанию, выпустившую некоторые из самых ранних электронных вычислительных машин BINAC (для военных приложений) и UNIVAC (для коммерческих приложений). Вновь Бюро переписи США сыграло в этом свою роль, приобретя первый UNIVAC. В это время далеко не всем было ясно, что ЭВМ станут экономически выгодными: вычислительные машины могли сортировать быстрее, но они и стоили дороже. Поэтому программисты UNIVAC под руководством Фрэнсис Э. Гольбертон (Frances E. Holberton) приложили значительные усилия к созданию программ внешней сортировки, работающих с высокой скоростью; их первые программы повлияли также на разработку оборудования. По их оценкам 100 млн записей по 10 слов могли быть рассортированы на UNIVAC за 9 000 ч (т. е. за 375 дней).

Вычислительная машина UNIVAC I, официально запущенная в эксплуатацию в июле 1951 года, имела оперативную память объемом 1 000 12-буквенных (72-битовых) слов. В ней предусматривались чтение и запись на ленту блоков по 60 слов со скоростью 500 слов в секунду; чтение могло быть прямым или обратным, допускалось совмещение операций чтения, записи и вычислений. В 1948 году миссис Гольбертон придумала интересный способ выполнения двухпутевого слияния с полным совмещением чтения, записи и вычислений с использованием шести буферов ввода. Для каждого вводного файла имелись один "текущий" буфер и два "вспомогательных" буфера; она предложила так организовать слияние, что всякий раз, когда приходило время вывода одного блока, два текущих буфера ввода содержали вместе один блок необработанных данных. Следовательно, за время формирования

каждого выводного блока ровно один буфер ввода становился пустым и можно было организовать работу так, чтобы три вспомогательных буфера из четырех оказывались заполненными всякий раз, когда данные читались в оставшийся буфер. Этот метод чуть быстрее метода прогнозирования (см. алгоритм 5.4.6F), так как нет необходимости проверять результат одного ввода перед началом следующего. [См. *Collation Methods for the UNIVAC System* (Eckert-Mauchly Computer Corp., 1950), 2 volumes.]

Кульминацией этой работы стало создание генератора программ сортировки, который был первой крупной программой, разработанной для автоматического программирования. Пользователь указывал размер записи, позиции ключей (до пяти) в частичных полях каждой записи и "концевые" ключи, отмечающие конец файла, и после этого генератор сортировки порождал требуемую программу сортировки для файлов на одной бобине. На первом проходе этой программы выполнялась внутренняя сортировка блоков по 60 слов с использованием метода сравнения и подсчета (алгоритм 5.2C); затем выполнялось несколько сбалансированных двухпутевых проходов слияния с обратным чтением, исключающих сцепление лент, как описано выше. [См. *Master Generating Routine for 2-way Sorting* (Eckert-Mauchly Division of Remington Rand, 1952). Первый набросок этого доклада был озаглавлен "Основная составляющая программа двухпутевого слияния" (*Master Prefabrication Routine for 2-way Collation*)! См. также F. E. Holberton, *Symposium on Automatic Programming* (Office of Naval Research, 1954), 34–39.]

К 1952 году многие методы внутренней сортировки прочно вошли в программистский фольклор, но теория была развита сравнительно слабо. Даниэль Гольденберг (Daniel Goldenberg) [*Time analyses of various methods of sorting data*, Digital Computer Laboratory memo M-1680 (Mass. Inst. of Tech., 17 October, 1952)] запрограммировал для машины Whirlwind computer пять различных методов и выполнил анализ наилучшего и наихудшего случаев для каждой программы. Он нашел, что для сортировки сотни 15-битовых записей по 8-битовому ключу наилучшие по скорости результаты получаются в том случае, если используется таблица из 256 слов и каждая запись помещается в единственную соответствующую ее ключу позицию, а затем эта таблица сжимается. Однако данный метод имел очевидный недостаток: запись уничтожалась, если последующая имела тот же ключ. Остальные четыре проанализированных метода были ранжированы следующим образом: прямое двухпутевое слияние лучше поразрядной сортировки с основанием 2, которая лучше простого выбора, который, в свою очередь, лучше метода пузырька.

Эти результаты получили дальнейшее развитие в диссертации Гарольда Х. Сьюворда (Harold H. Seward) в 1954 году [*Information sorting in the application of electronic digital computers to business operations*, Digital Computer Lab. report R-232 (Mass. Inst. of Tech., 24 May, 1954; 60 pages)]. Сьюворд высказал идеи распределяющего подсчета и выбора с замещением. Он показал, что первый отрезок случайной перестановки имеет среднюю длину $e - 1$, и проанализировал наряду с методами внутренней сортировки методы внешней сортировки, причем не только на магнитных лентах, но и на устройствах внешней памяти других типов.

Еще более достойная внимания диссертация — на этот раз докторская — была написана Говардом Б. Демутом (Howard B. Demuth) в 1956 году [*Electronic Data Sorting* (Stanford University, October, 1956), 92 pages; *IEEE Trans.* **C-34** (1985),

296–310]. Эта работа помогла заложить основы теории сложности вычислений. В ней рассматривались три абстрактные модели задачи сортировки: с использованием циклической памяти, линейной памяти и памяти с произвольным доступом; для каждой модели были разработаны оптимальные или почти оптимальные методы. (См. упр. 5.3.4-68.) Хотя непосредственно из диссертации Демута не вытекает никаких практических следствий, в ней содержатся важные идеи о том, как связать теорию с практикой.

Таким образом, история решения проблемы сортировки тесно связана с большинством этапов развития вычислительной техники: с первыми машинами для обработки данных, первыми хранимыми программами, первым программным продуктом, первыми методами буферизации, первой работой по анализу алгоритмов и сложности вычислений.

Ни один из документов, касающихся ЭВМ и упомянутых выше, не появлялся в "открытой литературе". Так уж случилось, что почти вся ранняя история вычислительных машин отражена в сравнительно труднодоступных докладах, поскольку относительно немного людей было в то время связано с вычислительной техникой. Наконец, в 1955 и 1956 годах литература о сортировке проникает в печать в виде трех больших обзорных статей. Первая статья была подготовлена Дж. К. Хоскеном (J. C. Hosken) [*Proc. Eastern Joint Computer Conference* **8** (1955), 39–55]. Он начинает с тонкого наблюдения: "Чтобы снизить стоимость единицы результата, люди обычно укрупняют операции. Но при этом стоимость единицы сортировки не уменьшается, а возрастает". Хоскен описал все оборудование специального назначения, имевшееся в продаже, а также методы сортировки с использованием существовавших на то время ЭВМ. Его библиография включает 54 ссылки и основана большей частью на брошюрах фирм-изготовителей.

Подробная статья Э. Г. Френда (E. H. Friend) *Sorting on Electronic Computer Systems* [*JACM* **3** (1956), 134–168] явилась важной вехой в развитии технологии сортировки. Хотя за время, прошедшее с 1956 года, было разработано множество методов, эта статья все еще необычно современна во многих отношениях. Фрэнд дал тщательное описание весьма большого числа алгоритмов внутренней и внешней сортировки и обратил особое внимание на методы буферизации и характеристики накопителей на магнитных лентах. Он предложил некоторые новые методы (например, выбор из дерева, метод двуглавого змия и прогнозирования) и проанализировал некоторые математические свойства старых методов.

Третий обзор по сортировке, который появился в то время, был подготовлен Д. У. Дэвисом (D. W. Davies) [*Proc. Inst. Elect. Engineers* **103B**, Supplement 1 (1956), 87–93]. В последующие годы было опубликовано еще несколько прекрасных обзоров: D. A. Bell [*Comp. J.* **1** (1958), 71–77], A. S. Douglas [*Comp. J.* **2** (1959), 1–9]; D. D. McCracken, H. Weiss, T. Lee [*Programming Business Computers* (New York: Wiley, 1959), Chapter 15, pages 298–332], I. Flores [*JACM* **8** (1961), 41–80], K. E. Iverson [*A Programming Language* (New York: Wiley, 1962), Chapter 6, 176–245], C. C. Gotlieb [*CACM* **6** (1963), 194–201], T. N. Hibbard [*CACM* **6** (1963), 206–213], M. A. Goetz [*Digital Computer User's Handbook*, edited by M. Klerer and G. A. Korn (New York: McGraw-Hill, 1967), Chapter 1.10, pages 1.292–1.320]. В ноябре 1962 года ACM организовала симпозиум по сортировке (бóльшая часть работ, представленных на этом симпозиуме, опубликована в мае 1963 года в выпуске *CACM*). Они

дают хорошее представление о состоянии работ в этой области на то время. Обзор К. К. Готлиба (C. C. Gotlieb) о современных генераторах сортировки, обзор Т. Н. Хиббарда (T. N. Hibbard) о внутренней сортировке с минимальной памятью и раннее исследование Дж. А. Хаббарда (G. U. Hubbard) о сортировке файлов на дисках — статьи из этого сборника, заслуживающие наибольшего внимания.

За прошедший период были открыты новые методы сортировки: вычисление адреса (1956), слияние с вставкой (1959), обменная поразрядная сортировка (1959), каскадное слияние (1959), метод Шелла с убывающим смещением (1959), многофазное слияние (1960), вставки в дерево (1960), осциллирующая сортировка (1962), быстрая сортировка Хоара (1962), пирамидальная сортировка Уильямса (1964), обменная сортировка со слиянием Бэтчера (1964). История каждого отдельного алгоритма прослеживается в тех разделах настоящей главы, в которых этот метод описывается. Конец 60-х годов нашего столетия ознаменовался интенсивным развитием соответствующей теории.

Полная библиография всех работ, изученных автором при написании и подготовке первого издания этой главы и составленная с помощью Р. Л. Ривест (R. L. Rivest), приводится в *Computing Reviews* **13** (1972), 283–289.

Последние достижения. Со времени выхода из печати первого издания этой книги (1970 г.) появилось много сообщений об изобретении новых алгоритмов сортировки, однако почти все они являются вариациями уже известных алгоритмов. *Быстрая сортировка на множестве ключей*, которая обсуждалась в упр. 5.2.2–30, представляет собой прекрасный пример таких более новых методов.

Другая тенденция в этой области, представляющая, скорее, теоретический интерес, — изучение *адаптивных* схем сортировки. Такие схемы по замыслу разработчиков должны гарантировать более быстрое выполнение сортировки в случаях, когда входная последовательность удовлетворяет какому-либо из заранее установленных критериев. [См., например, H. Mannila, *IEEE Transactions* **C-34** (1985), 318–325; V. Estivill-Castro, D. Wood, *Computing Surveys* **24** (1992), 441–476; C. Levcopoulos, O. Petersson, *Journal of Algorithms* **14** (1993), 395–413; A. Moffat, G. Eddy, O. Petersson, *Software Practice & Experience* **26** (1996), 781–797.]

Разительные изменения в характеристиках аппаратного обеспечения современных компьютерных систем стимулировали интерес к исследованию проблемы эффективности алгоритмов сортировки при совершенно новых критериях стоимости затрат. [См., например, обсуждение применения виртуальной памяти в упр. 5.4.9–20.] Эффект от применения аппаратной кэш-памяти при внутренней сортировке анализировался в работе A. LaMarca, R. E. Ladner, *J. Algorithms* **31** (1999), 66–104. Ее авторы пришли к выводу, что не следует включать шаг Q9 в алгоритм 5.2.2Q, если для сортировки используется компьютер с современной архитектурой, хотя для компьютеров традиционной структуры, каковым является наш `MIX`, алгоритм в прежнем виде вполне работоспособен. Вместо того чтобы завершать быструю сортировку с помощью метода прямых вставок, предлагается, что, по мнению авторов, значительно лучше, заранее сортировать короткие подмассивы, сохраняя их ключи в кэш-памяти.

А что можно сказать о нынешнем состоянии в области сортировки больших массивов данных? Одним из распространенных тестов для сравнительной оценки

различных средств решения такого рода задач с 1985 года стала сортировка миллиона 100-символьных записей с равномерно распределенными случайными 10-символьными ключами. Предполагается, что исходная последовательность и результат должны храниться на диске, а целью является минимизация времени обработки, включая время загрузки и запуска программы. В работе R. C. Agarwal, *SIGMOD Record* **25**, 2 (June, 1996), 240–246, описан эксперимент, проведенный на компьютере IBM RS/6000, модель 39H, в котором используется процессор с RISC-архитектурой. Методом поразрядной сортировки обрабатывались файлы, распределенные между восемью дисками, и задача была решена за 5.1 с. В подобных экспериментах узким местом является скорость ввода-вывода. Процессору для решения этой задачи понадобилось всего 0.6 с! Можно получить еще более высокую скорость, если параллельно использовать несколько процессоров. Сеть из 32 рабочих станций UltraSPARC I, каждая из которых была оснащена двумя собственными дисками, может сортировать миллион записей за 2.41 с, используя гибридный метод, названный NOW-Sort [A. C. Arpaci-Dusseau, R. H. Arpaci-Dusseau, D. E. Culler, J. M. Hellerstein, D. A. Patterson, *SIGMOD Record* **26**, (June, 1997), 243–254].

Изложенное выше подводит нас к выводу, что предлагаемый тест сортировки миллиона записей является, скорее, оценкой времени ввода-вывода, чем эффективности метода собственно сортировки; входные последовательности бóльшей длины потребуют других, более "значимых", тестов. Например, последовательность поистине глобального масштаба для *терабайтовой сортировки* — 10^{10} записей по 100 символов — была рассортирована за 2.5 ч. Этот результат получен в сентябре 1997 года на системе Silicon Graphics Origin2000, включающей 32 процессора, 8 Гбайт оперативной памяти и 559 дисков по 4 Гбайт. Массив был рассортирован с помощью доступной на рынке программного обеспечения программы Nsort™, разработанной К. Нибергом (C. Nyberg), Ч. Койстером (C. Koester) и Дж. Греем (J. Gray), в которой используется неопубликованный до сих пор метод обработки.

Но даже тест сортировки терабайтовых массивов может по нынешним временам оказаться недостаточно емким показателем. Наилучший из имеющихся на сегодняшний день претендентов на звание "универсального" теста эффективности сортировки, который обещает жить вечно (во всяком случае, так нам хотелось бы), — это так называемый критерий *Минут-Сорт*. Суть его состоит в выяснении, сколько 100-символьных записей можно рассортировать за 60 с. Когда данная книга была на последней стадии готовности к печати, рекордсменом по этому критерию был метод NOW-Sort; 30 марта 1997 года 95 рабочим станциям, объединенным в сеть, понадобилось всего 59.21 с на сортировку 90.25 млн записей. Но и результаты, полученные на самых современных системах, не опровергли ни одну из основополагающих оценок, полученных теоретически.

Подводя итог, можно с уверенностью заявить, что проблема эффективности сортировки остается сегодня такой же злободневной, как и ранее.

УПРАЖНЕНИЯ

1. [*05*] Подведите итог этой главе; сформулируйте обобщение теоремы 5.4.6A.

2. [*20*] Взяв за основу табл. 1, скажите, какой из методов сортировки списков с шестиразрядными ключами будет наилучшим для машины MIX.

3. [*37*] (*Устойчивая сортировка с минимальным объемом памяти.*) Считается, что алгоритм сортировки требует минимальной памяти, если он использует для своих переменных только $O((\log N)^2)$ бит памяти сверх пространства, необходимого для размещения N записей. Алгоритм должен быть общим в том смысле, что он должен работать при любых N, а не только при определенном значении N, если, конечно, предполагается, что при вызове алгоритма для сортировки обеспечивается достаточное количество памяти с произвольным доступом. Во многих изученных нами алгоритмах сортировки это требование минимальной памяти нарушается; в частности, запрещено использование N полей связи. Быстрая сортировка (алгоритм 5.2.2Q) удовлетворяет требованию минимальной памяти, но время выполнения в наихудшем случае пропорционально N^2. Пирамидальная сортировка (алгоритм 5.2.3H) является единственным среди изученных нами алгоритмом типа $O(N \log N)$, который использует минимальный объем памяти, хотя можно сформулировать еще один подобный алгоритм, если использовать идею из упр. 5.2.4–18.

Самым быстрым общим алгоритмом из изученных нами, который *устойчиво* сортирует ключи, является метод слияния списков (алгоритм 5.2.4L), однако он использует не минимальную память. Фактически единственными устойчивыми алгоритмами сортировки с минимальной памятью, которые мы анализировали, были методы типа $\Omega(N^2)$ (простые вставки, метод пузырька и вариации на тему простого выбора).

Разработайте устойчивый алгоритм сортировки с минимальной памятью, требующий менее $O(N(\log N)^2)$ машинных циклов в наихудшем случае. [*Указание.* Можно создать устойчивый метод слияния, удовлетворяющий критерию минимальной памяти, который затрачивает на сортировку $O(N \log N)$ машинных циклов.]

▶ **4.** [*28*] Алгоритм сортировки называется *бережливым*, если принимает решение только на основе сравнения ключей и никогда не выполняет тех сравнений, результат которых может быть предсказан на основе сравнений, выполненных ранее. Какие из перечисленных в табл. 1 методов являются бережливыми?

5. [*46*] Довольно сложно сравнивать неслучайные данные, в которых присутствует множество равных ключей. Придумайте тест для определения эффективности сортировки, который (i) был бы актуальным как сегодня, так и через 100 лет, (ii) не включал бы в рассмотрение случай равномерно распределенных случайных ключей и (iii) не использовал бы входных последовательностей данных, которые изменились бы со временем.

Я посчитал бы цель достигнутой, если бы мне удалось рассортировать и расположить в логическом порядке основную часть того огромного материала, касающегося сортировки, который появился за несколько последних лет.

— ДЖ. К. ХОСКЕН (J. C. HOSKEN) (1955)

ГЛАВА 6

ПОИСК

Взглянем на запись...

— ЭЛ СМИТ (AL SMITH) (1928)

ЭТА ГЛАВА могла бы носить более претенциозное название — "Хранение и получение информации"; с другой стороны, ее можно было бы назвать кратко и просто — "Просмотр таблиц". В ней мы займемся вопросами накопления информации в памяти компьютера и рассмотрим способы ее быстрого извлечения. Зачастую мы сталкиваемся с избыточной информацией, и лучшее, что с ней можно сделать, — забыть о ней или уничтожить ее. Однако нередки ситуации, когда крайне важно сохранить материал и организовать его таким образом, чтобы впоследствии обеспечить к нему максимально быстрый доступ.

Основная часть этой главы посвящена изучению простейшей задачи: поиску информации, сохраненной с конкретным идентификатором. Например, в численном приложении может понадобиться найти $f(x)$ по данному x и таблице значений f; другим примером может послужить поиск перевода на английский язык некоторого русского слова.

В целом, мы предполагаем, что имеется набор из N записей и задача состоит в нахождении одной из них. Как и в случае сортировки, мы полагаем, что каждая запись включает специальное поле, именуемое ее *ключом*; данный термин удачен, так как, несомненно, отражает тот печальный факт, что ежедневно миллионы людей тратят время и силы на поиски собственных неизвестно куда запропастившихся ключей... В общем случае нам требуется N различных ключей для того, чтобы каждый из них однозначно идентифицировал связанную с ним запись. Набор всех записей именуется *таблицей* или *файлом*, причем слово "таблица" обычно используется для описания маленького файла, а слово "файл" — для описания большой таблицы. Большой же файл (или группу файлов) часто называют *базой данных*.

Алгоритм поиска имеет так называемый *аргумент*, K, и задача заключается в нахождении записи, для которой K служит ключом. Результатом поиска может быть одно из двух: либо поиск завершился *успешно*, и уникальная запись, содержащая K, найдена, либо поиск оказался *неудачным*, и запись с ключом K не найдена. После неудачного поиска иногда желательно внести новую запись, содержащую K, в таблицу. Метод, осуществляющий это, называется алгоритмом *поиска и вставки*. Некоторые аппаратные устройства, известные как *ассоциативная память*, решают данную проблему автоматически, путем, который можно сравнить с функционированием мозга человека, однако мы все же будем изучать технологию поиска, применяемую в обычных цифровых компьютерах общего назначения.

Хотя цель поиска — нахождение информации, хранящейся в ассоциированной с K записи, алгоритмы в этой главе предназначены для поиска K. После нахождения K поиск связанной с ним информации становится тривиальной задачей, зависящей от способа хранения информации. Например, найдя K в TABLE $+ i$, мы обнаружим связанные с ним данные в TABLE $+ i + 1$, в DATA $+ i$ или в еще каком-то месте, которое определяется способом хранения информации. Поэтому мы считаем задачу выполненной в тот момент, когда находим K (или убеждаемся в его отсутствии).

Поиск обычно является наиболее "времяемкой" частью многих программ, и замена плохого метода поиска хорошим может значительно увеличить скорость работы программы. К тому же часто представляется возможным так изменить данные или используемые структуры, что поиска удается избежать совсем. Одним из таких способов может быть связь данных (например, при использовании списка с двойными связями поиск предшествующего и последующего элементов данного становится совершенно излишним). В случае, когда мы можем выбирать ключи, имеется еще один способ избежать поиска: выбрать в качестве ключа натуральные числа $\{1, 2, \ldots, N\}$; при этом запись с ключом K просто размещается в TABLE $+$ K. Обе эти технологии были использованы, чтобы избежать поиска в алгоритме топологической сортировки , обсуждавшемся в разделе 2.2.3. Однако поиск необходим, если объекты для топологической сортировки представлены не числовыми, а символьными значениями. Естественно, в этом случае очень важно подобрать эффективный алгоритм поиска.

Методы поиска могут быть классифицированы несколькими способами. Мы можем разделить их на внутренний и внешний поиск так же, как в главе 5 мы разделяли алгоритмы сортировки на внутренние и внешние. Возможно деление на статические и динамические методы поиска, где термин "статический" означает, что содержимое таблицы остается неизменным и главная задача — уменьшить время поиска без учета времени, необходимого для настройки таблицы. Термин "динамический" означает, что таблица часто изменяется путем вставки (а возможно, и удаления) элементов. Третья возможная схема классификации методов поиска — их разделение в зависимости от того, на чем они основаны: на сравнении ключей или на некоторых числовых свойствах ключей (по аналогии с разделением на сортировку методом сравнения и сортировку методом распределения). И наконец, можно разделить методы сортировки на методы с использованием ключей действительных и преобразованных (трансформированных).

Организация этой главы представляет собой комбинацию двух последних способов классификации. Раздел 6.1 посвящен методам последовательного поиска, т. е. методам поиска "в лоб". В разделе 6.2 рассмотрены усовершенствования, основанные на сравнении ключей с использованием алфавитного или числового порядка для управления решениями. Раздел 6.3 посвящен числовому поиску, а в разделе 6.4 обсуждается важный класс методов с общим названием "хеширование", основанных на арифметическом преобразовании действительных ключей. В каждом из этих разделов рассматривается как внутренний, так и внешний поиск (как для статического, так и для динамического случаев). В каждом разделе вы найдете описание достоинств и недостатков рассматриваемых алгоритмов.

Поиск и сортировка зачастую тесно связаны между собой. Например, рассмотрим следующую задачу. *Даны два числовых множества,* $A = \{a_1, a_2, \ldots, a_m\}$ *и* $B = \{b_1, b_2, \ldots, b_n\}$. *Необходимо определить, является ли множество* A *подмножеством множества* B:$A \subseteq B$. Очевидными представляются такие варианты решения.

1. Сравнивать каждое a_i со всеми b_j последовательно до нахождения совпадения.
2. Сначала рассортировать множества A и B, а затем сделать только один последовательный проход с проверкой по обоим файлам.
3. Внести все элементы b_j в таблицу и выполнить поиск каждого значения a_i.

Каждое из этих решений имеет свои достоинства для различных значений m и n. Решение 1 требует порядка c_1mn единиц времени, где c_1 — некоторая константа. Решение 2 требует порядка $c_2(m \lg m + n \lg n)$ единиц времени, где c_2 — некоторая (бóльшая) константа. При выборе подходящего метода хеширования решение 3 займет около $c_3m + c_4n$ единиц времени для некоторых (еще бóльших) констант c_3 и c_4. Отсюда следует, что решение 1 подходит для очень небольших значений m и n; при увеличении множеств лучшим становится решение 2. Затем наилучшим станет решение 3 — до тех пор, пока n не превысит размер внутренней памяти. После этого наилучшим обычно вновь становится решение 2, пока n не вырастет до совсем уж громадных значений... Итак, мы видим, что существуют ситуации, в которых сортировка служит хорошей заменой поиску, а поиск — отличной заменой сортировке.

Более сложные задачи поиска зачастую сводятся к более простым (которые мы и рассматриваем). Предположим, например, что ключи представляют собой слова, которые могут быть записаны с небольшими ошибками. Наша задача — найти запись, невзирая на ошибки в ключах. Если мы сделаем две копии файла, в одном из которых ключи расположены в обычном лексикографическом порядке, а в другом — в обратном порядке, справа налево (как если бы их читали задом наперед), искаженный аргумент поиска будет, вероятно, совпадать до половины (или более) своей длины с записью в одном из файлов. Методы поиска, описываемые в разделах 6.2 и 6.3, таким образом, могли бы быть приспособлены для поиска по искаженному ключу.

Подобные задачи привлекли внимание в связи с вводом в действие систем резервирования билетов на самолеты и других подобных им систем, в которых велика вероятность возникновения ошибки из-за плохой слышимости или некаллиграфического почерка. Целью исследований был поиск метода преобразования аргумента в некий код, который позволил бы группировать различные варианты одной фамилии. Далее описан метод "Soundex", первоначально разработанный Маргарет К. Оделл (Margaret K. Odell) и Робертом С. Расселом (Robert C. Russell) [см. *U. S. Patents 1261167* (1918), *1435663* (1922)] и нашедший широкое применение для кодирования фамилий.

1. Оставить первую букву имени и удалить все буквы a, e, h, i, o, u, w, y в других позициях.
2. Назначить оставшимся буквам (кроме первой) следующие числовые значения:

b, f, p, v → 1	l → 4
c, g, j, k, q, s, x, z → 2	m, n → 5
d, t → 3	r → 6

3. Если в исходном слове до выполнения шага 1 две или более буквы с одним и тем же кодом стояли рядом, удалить их все, кроме первой.
4. Преобразовать полученный результат в формат "буква, цифра, цифра, цифра" (приписывая необходимое количество нулей справа, если в результате получилось меньше трех цифр, или отбрасывая лишние цифры, если их больше трех).

Например, фамилии Euler, Gauss, Hilbert, Knuth, Lloyd, Łukasiewicz и Wachs имеют коды E460, G200, H416, K530, L300, L222 и W200 соответственно. Естественно, одинаковые коды могут иметь и совсем разные имена. Так, приведенные выше коды могут быть получены из следующих фамилий: Ellery, Ghosh, Heilbronn, Kant, Liddy, Lissajous и Waugh. С другой стороны, такие схожие имена, как Rogers и Rodgers, Sinclair и St. Clair или Tchebysheff и Chebyshev, имеют разные коды. Тем не менее коды Soundex существенно увеличивают вероятность нахождения имени по одному из вариантов написания. (Чтобы получить более детальную информацию по этому вопросу, обратитесь к работам C. P. Bourne, D. F. Ford, *JACM* **8** (1961), 538–552; Leon Davidson, *CACM* **5** (1962), 169–171; *Federal Population Censuses 1790–1890* (Washington, D.C.: National Archives, 1971), 90).

При использовании схем типа Soundex нет необходимости в предположении, что все ключи различны. Мы можем составить списки записей с одинаковыми кодами, рассматривая каждый список в качестве отдельного модуля.

В больших базах данных наблюдается тенденция к более сложным выборкам. Зачастую пользователи рассматривают различные поля записей как потенциальные ключи с возможностью поиска, если известна только часть информации, содержащейся в ключе. Например, имея большой файл с информацией об артистах, продюсер может захотеть найти незанятых актрис в возрасте от 25 до 30 лет, неплохо танцующих и говорящих с французским акцентом. Имея подобный файл с бейсбольной статистикой, спортивный обозреватель может захотеть узнать общее количество очков, заработанных командой Chicago White Sox в 1964 году в седьмых периодах ночных игр при условии, что подающий был левшой... Люди любят задавать сложные вопросы. Для того чтобы иметь возможность получить на них ответ, в книге имеется раздел 6.5, в котором содержится введение в методы поиска по вторичному ключу (многоатрибутный поиск).

Прежде чем перейти к собственно изучению методов поиска, полезно взглянуть на историю данного вопроса. В докомпьютерную эру имелось множество книг с таблицами логарифмов, тригонометрическими и другими таблицами (те, кто помнят, что означает аббревиатура Б3-21 или Б3-34, несомненно, помнят, что такое "таблицы Брадиса". — *Прим. перев.*). Фактически многие математические вычисления были сведены к поиску. Затем эти таблицы трансформировались в перфокарты, которые использовались для решения научных задач с помощью распознающих, сортирующих и копирующих перфораторных машин. Однако с появлением компьютеров с хранимыми программами стало очевидным, что проще вычислить значение $\log x$ или $\cos x$ заново, чем найти его в таблице. (Следует, однако, заметить, что на определенном этапе развития вычислительной техники для некоторых приложений, особо критичных ко времени расчетов (в основном для игр с графическим интерфейсом) оказалось крайне выгодным вернуться к предвычислению таблиц функций, а в процессе работы вместо вычислений осуществлять поиск. К тому же в подобных

случаях можно было использовать более быструю целочисленную арифметику. — *Прим. перев.*)

Хотя задача сортировки привлекала большое внимание еще на заре компьютерной эры, алгоритмы поиска оставались в забвении достаточно долгое время. Малая внутренняя память и наличие только устройств последовательного доступа (наподобие лент) для хранения больших файлов делали поиск либо тривиальным, либо невозможным.

Развитие и удешевление памяти с произвольным доступом уже в 50-х годах привело к пониманию того, что проблема поиска важна и интересна сама по себе. После многих лет жалоб на ограниченность пространства программисты столкнулись с таким изобилием, которое попросту не могли переварить и эффективно использовать.

Первые обзоры по проблеме поиска опубликованы А. И. Думи (A. I. Dumey), *Computers & Automation* **5**, 12 (December, 1956), 6–9; В. В. Петерсоном (W. W. Peterson), *IBM J. Research & Development* **1** (1957), 130–146; Э. Д. Бутом (A. D. Booth), *Information and Control* **1** (1958), 159–164; А. Ш. Дугласом (A. S. Douglas), *Comp. J.* **2** (1959), 1–9. Более подробный обзор, посвященный проблемам сортировки, был сделан несколько позже Кеннетом Ю. Айверсоном (Kenneth E. Iverson), *A Programming Language* (New York: Wiley, 1962), 133–158, и Вернером Букхольцем (Werner Buchholz), *IBM Systems J.* **2** (1963), 86–111.

В начале 60-х годов было разработано несколько новых процедур поиска, основанных на древовидных структурах (с ними мы встретимся немного позже). Исследования алгоритмов поиска ведутся и в настоящее время.

6.1. ПОСЛЕДОВАТЕЛЬНЫЙ ПОИСК

"Начать с начала и продолжать, пока не будет найден искомый ключ; затем остановиться." Эта последовательная процедура представляет собой очевидный путь поиска и может служить отличной отправной точкой для рассмотрения множества алгоритмов поиска, поскольку они основаны на последовательной процедуре. Мы увидим, что за простотой последовательного поиска скрывается ряд очень интересных, несмотря на их простоту, идей.

Вот более точная формулировка алгоритма.

Алгоритм S (*Последовательный поиск* (*Sequential search*)). Дана таблица записей $R_1, R_2, \ldots, R_N$ с ключами $K_1, K_2, \ldots, K_N$ соответственно. Алгоритм предназначен для поиска записи с заданным ключом K. Предполагается, что $N \geq 1$.

S1. [Инициализация.] Установить $i \leftarrow 1$.

S2. [Сравнение.] Если $K = K_i$, алгоритм заканчивается успешно.

S3. [Продвижение.] Увеличить i на 1.

S4. [Конец файла?] Если $i \leq N$, перейти к шагу S2. В противном случае алгоритм заканчивается неудачно. ▌

Обратите внимание на то, что этот алгоритм может завершиться успешно (искомый ключ найден) и неудачно (искомый ключ отсутствует). Данное утверждение справедливо для большинства алгоритмов, рассматриваемых в этой главе.

MIX-программа пишется очень просто.

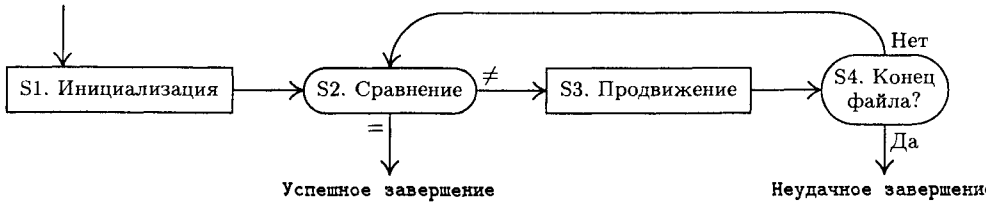


Рис. 1. Последовательный поиск.

Программа S (*Последовательный поиск*). Предположим, что K_i хранится по адресу `KEY` $+ i$, а оставшаяся часть записи (R_i) — по адресу `INFO` $+ i$. Программа использует rA $\equiv K$, rI1 $\equiv i - N$.

01	START	LDA	K	1	*S1. Инициализация.*
02		ENT1	1-N	1	$i \leftarrow 1$.
03	2H	CMPA	KEY+N,1	C	*S2. Сравнение.*
04		JE	SUCCESS	C	Выход, если $K = K_i$.
05		INC1	1	$C - S$	*S3. Продвижение.*
06		J1NP	2B	$C - S$	*S4. Конец файла?*
07	FAILURE	EQU	*	$1 - S$	Выход при отсутствии в таблице.

По адресу `SUCCESS` находится команда `LDA INFO+N,1`, которая помещает необходимую информацию в rA. ▌

Анализ этой программы несложен. Очевидно, что время выполнения алгоритма S зависит от двух факторов:

C — количество сравнений ключей;

$S = 1$ при успешном окончании поиска, 0 — при неудачном. (1)

Программа S требует для работы $5C - 2S + 3$ единиц времени. Если при поиске успешно найден $K = K_i$, получим $C = i$, $S = 1$; таким образом, полное время составляет $(5i + 1)u$. С другой стороны, если поиск неудачен, мы получим $C = N$, $S = 0$ и общее время работы программы — $(5N + 3)u$. Если все ключи поступают на вход программы с одинаковой вероятностью, то среднее значение C в случае успешного поиска равно

$$\frac{1 + 2 + \cdots + N}{N} = \frac{N + 1}{2}. \tag{2}$$

При этом значение среднеквадратичного отклонения, естественно, достаточно велико и составляет около $0.289N$ (см. упр. 1).

Данный алгоритм, несомненно, знаком всем программистам, но лишь некоторые из них знают, что этот способ — *не* самая лучшая реализация последовательного поиска! Небольшое изменение — и алгоритм выполняется существенно быстрее (если записей не слишком мало).

Алгоритм Q (*Быстрый последовательный поиск*). Перед вами тот же алгоритм, что и алгоритм S, однако в нем имеется дополнительное предположение о наличии фиктивной записи R_{N+1} в конце файла.

Q1. [Инициализация.] Установить $i \leftarrow 1$ и $K_{N+1} \leftarrow K$.

Q2. [Сравнение.] Если $K = K_i$, перейти к шагу Q4.

Q3. [Продвижение.] Увеличить i на 1 и перейти к шагу Q2.

Q4. [Конец файла?] Если $i \le N$, алгоритм заканчивается успешно; в противном случае алгоритм заканчивается неудачно ($i = N + 1$). ▮

Программа Q (*Быстрый последовательный поиск*). rA $\equiv K$, rI1 $\equiv i - N$.

```
01 START   LDA  K            1          Q1. Инициализация.
02         STA  KEY+N+1      1          K_{N+1} ← K.
03         ENT1 -N           1          i ← 0.
04         INC1 1          C+1-S        Q3. Продвижение.
05         CMPA KEY+N,1    C+1-S        Q2. Сравнение.
06         JNE  *-2        C+1-S        Переход к Q3, если K_i ≠ K.
07         J1NP SUCCESS      1          Q4. Конец файла?
08 FAILURE EQU  *          1-S         Выход при отсутствии в таблице. ▮
```

Используя те же значения C и S, что и в программе S, получим, что значение времени работы равно $(4C - 4S + 10)u$; таким образом, мы получаем выигрыш по сравнению с предыдущим алгоритмом в случае $C \ge 6$ для успешного поиска и $N \ge 8$ — для неудачного.

При переходе от алгоритма S к алгоритму Q использован важный ускоряющий принцип — при нескольких проверках во внутреннем цикле следует постараться свести их к одной.

Вот еще один способ сделать программу Q *еще* быстрее.

Программа Q′ (*Быстрый последовательный поиск*). rA $\equiv K$, rI1 $\equiv i - N$.

```
01 START   LDA  K              1            Q1. Инициализация.
02         STA  KEY+N+1        1            K_{N+1} ← K.
03         ENT1 -1-N           1            i ← -1.
04 3H      INC1 2          ⌊(C-S+2)/2⌋      Q3. Продвижение (дважды).
05         CMPA KEY+N,1    ⌊(C-S+2)/2⌋      Q2. Сравнение.
06         JE   4F         ⌊(C-S+2)/2⌋      Переход к шагу Q4, если K = K_i.
07         CMPA KEY+N+1,1  ⌊(C-S+1)/2⌋      Q2. Сравнение (следующее).
08         JNE  3B         ⌊(C-S+1)/2⌋      Переход к шагу Q3, если K ≠ K_{i+1}.
09         INC1 1          (C-S) mod 2      Продвижение i.
10 4H      J1NP SUCCESS        1            Q4. Конец файла?
11 FAILURE EQU  *              1-S          Выход при отсутствии в таблице. ▮
```

Внутренний цикл дублирован, что позволяет избежать выполнения половины инструкций "$i \leftarrow i + 1$", тем самым снижая затраты времени до

$$3.5C - 3.5S + 10 + \frac{(C - S) \bmod 2}{2}$$

единиц времени. При работе с большими таблицами это сохраняет до 30% нашего времени. Такой способ ускорения применим ко многим существующим программам на языках высокого уровня [см., например, D. E. Knuth, *Computing Surveys* **6** (1974), 266–269].

Можно воспользоваться другим улучшенным алгоритмом, если ключи расположены в порядке возрастания:

Алгоритм Т (*Последовательный поиск в упорядоченной таблице*). Дана таблица записей $R_1, R_2, \ldots, R_N$, ключи которых расположены в порядке возрастания: $K_1 < K_2 < \cdots < K_N$. Алгоритм предназначен для поиска записи с заданным ключом K. Для удобства и ускорения работы алгоритма предполагается наличие фиктивной записи R_{N+1} с ключом $K_{N+1} = \infty > K$.

Т1. [Инициализация.] Установить $i \leftarrow 1$.

Т2. [Сравнение.] Если $K \leq K_i$, перейти к шагу Т4.

Т3. [Продвижение.] Увеличить i на 1 и перейти к шагу Т2.

Т4. [Равенство?] Если $K = K_i$, алгоритм заканчивается успешно. В противном случае — неудачное завершение алгоритма. ▌

В предположении, что все входные аргументы-ключи равновероятны, алгоритм по скорости работы в случае успешного поиска аналогичен алгоритму Q; при неудачном же поиске отсутствие нужного ключа определяется примерно вдвое быстрее.

Во всех приведенных здесь алгоритмах использовалась запись с индексами для элементов таблиц (она более удобна для описания алгоритмов). Однако все описанные алгоритмы применимы и к другим типам данных, например к таблицам со *связанным* представлением данных, поскольку в них данные также расположены последовательно (см. упр. 2–4).

Частота обращений. До сих пор мы предполагали, что все аргументы поиска равновероятны. В общем случае это не так: вероятность запроса на поиск с ключом K_j равна p_j, причем $p_1 + p_2 + \cdots + p_N = 1$. Время, требуемое для успешного завершения поиска, пропорционально количеству сравнений C, которое имеет среднее значение

$$\bar{C}_N = p_1 + 2p_2 + \cdots + Np_N. \tag{3}$$

Если мы можем размещать записи в таблице в любом порядке, значение $\bar{C}_N$ минимально при

$$p_1 \geq p_2 \geq \cdots \geq p_N, \tag{4}$$

т. е. когда наиболее часто используемые записи находятся в начале таблицы.

Рассмотрим случаи различных распределений вероятностей и выясним, какой выигрыш может дать оптимальное расположение записей в таблице, указанное в (4). В случае равновероятного появления ключей $p_1 = p_2 = \cdots = p_N = 1/N$ формула (3) сводится к $\bar{C}_N = (N+1)/2$, т. е. к ранее полученному нами результату (2). Теперь предположим, что распределение вероятностей имеет вид

$$p_1 = \frac{1}{2}, \qquad p_2 = \frac{1}{4}, \qquad \ldots, \qquad p_{N-1} = \frac{1}{2^{N-1}}, \qquad p_N = \frac{1}{2^{N-1}}. \tag{5}$$

Согласно упр. 7 в данном случае $\bar{C}_N = 2 - 2^{1-N}$; при этом среднее количество сравнений, если записи расположены в надлежащем порядке, *меньше двух*.

Еще одно, клиновидное, распределение вероятностей определяется как

$$p_1 = Nc, \quad p_2 = (N-1)c, \quad \ldots, \quad p_N = c, \tag{6}$$

где $c = \frac{2}{N(N+1)}$. Здесь мы не получим такого эффекта, как при распределении (5). В случае клиновидного распределения

$$\bar{C}_N = c\sum_{k=1}^{N} k(N+1-k) = \frac{N+2}{3}, \tag{7}$$

и при оптимальном размещении записей экономится около трети времени, которое уходит на поиск, по сравнению со временем, необходимым при случайном размещении записей*.

Естественно, распределения (5) и (6) сугубо искусственны и не могут служить хорошим приближением реальных примеров. Более типично для реальных ситуаций распределение Зипфа:

$$p_1 = c/1, \quad p_2 = c/2, \quad \ldots, \quad p_N = c/N, \tag{8}$$

где $c = 1/H_N$. Оно получило известность благодаря работам Д. К. Зипфа (G. K. Zipf), который, исследуя естественные языки, обнаружил, что n-е по частоте употребления слово языка встречается с частотой, примерно обратно пропорциональной n [см. *The Psycho-Biology of Language* (Boston, Mass.: Houghton Mifflin, 1935); *Human Behavior and the Principle of Least Effort* (Reading, Mass.: Addison-Wesley, 1949)]. Аналогичные распределения встречаются, например, в таблицах переписи населения, в которых районы расположены в порядке убывания численности населения. В случае подчинения ключей в таблице закону Зипфа находим

$$\bar{C}_N = N/H_N. \tag{9}$$

Поиск в таком файле осуществляется примерно в $\frac{1}{2}\ln N$ раз быстрее, чем в неупорядоченном файле [см. A. D. Booth, L. Brandwood, and J. P. Cleave, *Mechanical Resolution of Linguistic Problems* (New York: Academic Press, 1958), 79].

Другое близкое к реальному распределение — это распределение, соответствующее правилу "80–20", которое часто наблюдается в коммерческих приложениях [см., например, W. P. Heising, *IBM Systems J.* **2** (1963), 114–115]. Это правило гласит, что 80% транзакций работают с 20% файла. Оно фрактально, т. е. оно применимо и к активным 20% файла; следовательно, 64% транзакций работают с 4% файла и т. д. Другими словами,

$$\frac{p_1 + p_2 + \cdots + p_{.20n}}{p_1 + p_2 + p_3 + \cdots + p_n} \approx .80 \tag{10}$$

для всех n. Вот одно из точно удовлетворяющих правилу распределений (при n, кратных 5):

$$p_1 = c, \quad p_2 = (2^\theta - 1)c, \quad p_3 = (3^\theta - 2^\theta)c, \quad \ldots, \quad p_N = \left(N^\theta - (N-1)^\theta\right)c, \tag{11}$$

где

$$c = 1/N^\theta, \qquad \theta = \frac{\log .80}{\log .20} = 0.1386. \tag{12}$$

Как вы можете убедиться, в этом случае для всех n $p_1 + p_2 + \cdots + p_n = cn^\theta$. Вероятности из (11) не очень удобны для работы; однако $n^\theta - (n-1)^\theta = \theta n^{\theta-1}\left(1 + O(1/n)\right)$,

* Имеется в виду случай равновероятного появления ключей. — *Прим. ред.*

и поэтому можно воспользоваться более простым распределением, приближенно удовлетворяющим правилу "80–20":

$$p_1 = c/1^{1-\theta}, \quad p_2 = c/2^{1-\theta}, \quad \ldots, \quad p_N = c/N^{1-\theta}, \tag{13}$$

где $c = 1/H_N^{(1-\theta)}$. Здесь, как и ранее, $\theta = \log .80/\log .20$, а $H_N^{(s)}$ — N-е гармоническое число порядка s, а именно — $1^{-s} + 2^{-s} + \cdots + N^{-s}$. Обратите внимание на схожесть этого распределения вероятности с законом Зипфа (8); с изменением θ от 1 до 0 закон распределения изменяется от равномерного к закону Зипфа. Применяя формулу (3) к распределению (13), получим среднее число сравнений для закона "80–20" (см. упр. 8):

$$\bar{C}_N = H_N^{(-\theta)}/H_N^{(1-\theta)} = \frac{\theta N}{\theta+1} + O(N^{1-\theta}) \approx 0.122N. \tag{14}$$

Изучая частоту употребления слов, Ю. С. Шварц (E. S. Schwartz) (см. интересный график на с. 422 в *JACM* **10** (1963)) предложил использовать в качестве более точного приближения распределение (13) с небольшими *отрицательными* значениями θ. Тогда значение

$$\bar{C}_N = H_N^{(-\theta)}/H_N^{(1-\theta)} = \frac{N^{1+\theta}}{(1+\theta)\zeta(1-\theta)} + O(N^{1+2\theta}) \tag{15}$$

получается существенно меньше, чем в случае (9) при $N \to \infty$.

Распределения, подобные (11) и (13), были впервые изучены Вильфредо Парето (Vilfredo Pareto) в связи с распределением богатства людей [*Cours d'Économie Politique* **2** (Lausanne: Rouge, 1897), 304–312]. Пусть p_k пропорционально состоянию k-го по богатству индивидуума, а p_N — состоянию более бедного индивидуума, занимающего в списке богатых N-е место. Тогда k/N представляет собой вероятность того, что состояние более богатого человека не менее чем в $x = p_k/p_N$ раз превосходит состояние более бедного. Таким образом, при $p_k = ck^{\theta-1}$ и $x = (k/N)^{\theta-1}$ описанная вероятность равна $x^{-1/(1-\theta)}$. Такое распределение в настоящее время называется *распределением Парето* с параметром $1/(1-\theta)$.

Любопытно, что Парето не понимал сути собственного распределения; он полагал, что значение θ, близкое к 0, соответствует более уравнительному обществу, чем значение, близкое к 1! Его ошибка была исправлена Коррадо Жини (Corrado Gini) [*Atti della III Riunione della Società Italiana per il Progresso delle Scienze* (1910), переиздана в его *Memorie di Metodologia Statistica* **1** (Rome, 1955), 3–120]. Жино был первым человеком, сформулировавшим и объяснившим важность соотношений, подобных закону "80–20" (10). Люди, как правило, не понимают сути таких распределений; они часто говорят о законе "75–25" или "90–10", как если бы главное, что придает смысл закону, заключалось в том, что в законе "a–b" выполняется равенство $a + b = 100$. Однако, как можно убедиться из (12), сумма $80 + 20$ здесь ни при чем...

Еще одно дискретное распределение, аналогичное (11) и (13), было предложено Д. Удни Юлом (G. Udny Yule) при изучении увеличения со временем количества биологических видов при различных моделях эволюции [*Philos. Trans.* **B213** (1924), 21–87]. Распределение Юла допускает значения $\theta < 2$:

$$p_1 = c, \; p_2 = \frac{c}{2-\theta}, \; p_3 = \frac{2c}{(3-\theta)(2-\theta)}, \; \ldots, \; p_N = \frac{(N-1)!\,c}{(N-\theta)\ldots(2-\theta)} = \frac{c}{\binom{N-\theta}{N-1}};$$

$$c = \frac{\theta}{1-\theta}\,\frac{\binom{N-\theta}{N}}{1-\binom{N-\theta}{N}}. \tag{16}$$

Граничные значения $c = 1/H_N$ и $c = 1/N$ получаются при $\theta = 0$ и $\theta = 1$.

Имеется еще одна часто цитируемая работа, однако ее популярность связана не с ее важностью, а лишь с тем, что это первая работа американского автора на эту тему [Alfred J. Lotka, *J Washington Academy of Sciences* **16** (1926), 317–323].

"Самоорганизующийся" файл. Приведенные вычисления вероятностей хороши, однако в большинстве случаев распределение вероятностей априори не известно. Можно было бы хранить в каждой записи счетчик обращений к ней и на основании полученных показаний переупорядочивать записи. Конечно, во многих ситуациях, как мы видели, такое переупорядочение приведет к значительной экономии времени; тем не менее зачастую не стоит выделять память для счетчиков — гораздо разумнее использовать ее, например, для технологии непоследовательного поиска, о чем будет рассказано ниже в данной главе.

Существует используемая многие годы простая схема, происхождение которой, увы, не известно. Эта схема позволяет получить неплохие результаты без привлечения счетчиков: когда запись успешно обнаружена, она перемещается в начало таблицы.

Идея этой "самоорганизующейся" технологии заключается в том, что наиболее часто используемые записи в результате будут располагаться в начале таблицы. Предположим, что N ключей встречаются среди аргументов поиска с вероятностями $\{p_1, p_2, \ldots, p_N\}$ и при этом каждый поиск совершается *независимо* от других. В таком предположении можно показать, что среднее количество сравнений, необходимых для поиска записи в таком самоорганизующемся файле стремится к предельному значению (см. упр. 11):

$$\widetilde{C}_N = 1 + 2\sum_{1\le i<j\le N} \frac{p_i p_j}{p_i + p_j} = \frac{1}{2} + \sum_{i,j} \frac{p_i p_j}{p_i + p_j}. \tag{17}$$

Например, если $p_i = 1/N$ при $1 \le i \le N$, самоорганизующаяся таблица будет находиться в неупорядоченном состоянии, а приведенная формула сведется к хорошо известному нам значению $(N+1)/2$. В общем случае полученное в (17) значение всегда меньше удвоенного оптимального значения из (3), так как $\widetilde{C}_N \le 1 + 2\sum_{j=1}^{N}(j-1)p_j = 2\bar{C}_N - 1$. В действительности $\widetilde{C}_N$ всегда меньше, чем $(\pi/2)\cdot\bar{C}_N$ [Chung, Hajela, and Seymour, *J. Comp. Syst. Sci.* **36** (1988), 148–157]. Эту константу нельзя улучшить, так как при p_j, пропорциональных $1/j^2$, достигается равенство.

Посмотрим, насколько хорошо этот метод работает при распределении вероятностей по закону Зипфа (8). В этом случае мы имеем:

$$\begin{aligned}\widetilde{C}_N &= \frac{1}{2} + \sum_{1\le i,j\le N} \frac{(c/i)(c/j)}{c/i + c/j} = \frac{1}{2} + c\sum_{1\le i,j\le N} \frac{1}{i+j}\\ &= \frac{1}{2} + c\sum_{i=1}^{N}(H_{N+i} - H_i) = \frac{1}{2} + c\sum_{i=1}^{2N} H_i - 2c\sum_{i=1}^{N} H_i\\ &= \tfrac{1}{2} + c\big((2N+1)H_{2N} - 2N - 2(N+1)H_N + 2N\big)\end{aligned}$$

$$= \tfrac{1}{2} + c\big(N \ln 4 - \ln N + O(1)\big) \approx 2N/\lg N \tag{18}$$

(см. формулы 1.2.7–(8) и 1.2.7–(3)). Полученная величина существенно лучше, чем $\frac{1}{2}N$ при достаточно больших N, и только в $\ln 4 \approx 1.386$ раз превышает количество сравнений при оптимальном размещении записей (см. (9)).

Вычислительные эксперименты с реальными таблицами символов компиляторов показали, что зачастую метод самоорганизации работает даже лучше, чем предсказывается; это связано с тем, что последовательные успешные поиски не являются независимыми и небольшие группы ключей зачастую вместе участвуют в поиске.

Впервые такая самоорганизующаяся схема была исследована Джоном Мак-Кэйбом (John McCabe) [*Operations Research* **13** (1965), 609–618], который и получил соотношение (17).

Мак-Кэйб предложил также другую интересную схему, при которой успешно обнаруженный ключ, не находящийся в начале таблицы, просто *меняется местами с предыдущим*, а не перемещается в начало таблицы. Он также установил, что предельное среднее время поиска для этого метода, в предположении независимости поисков, не превышает значения из (17). Несколькими годами позже Рональд Л. Ривест (Ronald L. Rivest) доказал, что метод перестановки при длительной работе использует строго меньше сравнений, чем метод перемещения в начало таблицы — естественно, исключая случаи, когда $N \le 2$ или когда все ненулевые вероятности равны [*CACM* **19** (1976), 63–67]. Однако переход к асимптотическому пределу в этом случае происходит более медленно, чем при перемещении записей в начало таблицы [J. R. Bitner, *SICOMP* **8** (1979), 82–110]. Более того, Дж. Л. Бентли (J. L. Bentley), К. К. Мак-Геч (C. C. McGeoch), Д. Д. Слитор (D. D. Sleator) и Р. Е. Таржан (R. E. Tarjan) доказали, что при методе перемещения в начало таблицы количество обращений к памяти никогда не превысит более чем в четыре раза это количество для любого алгоритма при работе с линейными списками, заданного любой последовательностью обращений; методы же подсчета частот и перестановки таким свойством не обладают [*CACM* **28** (1985), 202–208, 404–411]. См. также *SODA* **8** (1997), 53–62, где приведены интересные результаты эмпирического изучения более 40 эвристических методов самоорганизующихся списков, проведенных Р. Бачрачем (R. Bachrach) и Р. Эль-Янивом (R. El-Yaniv).

Поиск на ленте с неравными записями. Рассмотрим теперь несколько иную задачу. Предположим, что таблица, в которой проводится поиск, хранится на ленте и при этом отдельные записи имеют различную длину. Примером такого хранения информации может служить лента системной библиотеки старых операционных систем. Стандартные системные программы (например такие, как компиляторы, ассемблеры, загружаемые подпрограммы и генераторы отчетов) являются "записями" на этой ленте, и большинство пользовательских заданий должно начинать выполнение с поиска необходимого программного обеспечения. Такая постановка задачи делает неприемлемым анализ алгоритма S, поскольку шаг S3 теперь выполняется за разные промежутки времени для разных записей. Поэтому мы не можем ограничиться в нашем анализе только количеством сравнений.

Пусть L_i — длина записи R_i и пусть p_i — вероятность того, что выполняется поиск именно этой записи. В таком случае среднее время поиска примерно

пропорционально

$$p_1L_1 + p_2(L_1 + L_2) + \cdots + p_N(L_1 + L_2 + L_3 + \cdots + L_N). \tag{19}$$

При $L_1 = L_2 = \cdots = L_N = 1$ это выражение сводится к уже исследованному нами случаю (3).

Представляется разумным и логичным разместить записи, обращения к которым происходят чаще, чем к другим, в начале ленты, однако в данном случае это вовсе не такая хорошая мысль, как кажется! Например, предположим, что у нас есть лента с двумя программами — A и B, причем обращения к A происходят в два раза чаще, чем к B, но при этом она в четыре раза длиннее. Тогда $N = 2$, $p_A = \frac{2}{3}$, $L_A = 4$, $p_B = \frac{1}{3}$, $L_B = 1$. Если мы поместим на ленту сначала A, а затем B, руководствуясь описанной "логикой", среднее время поиска станет равным $\frac{2}{3} \cdot 4 + \frac{1}{3} \cdot 5 = \frac{13}{3}$. Если же воспользоваться "нелогичным" решением, поместив на ленту сначала B, а затем A, среднее время поиска сократится до $\frac{1}{3} \cdot 1 + \frac{2}{3} \cdot 5 = \frac{11}{3}$.

Оптимальное расположение информации на ленте (с точки зрения скорости поиска) можно определить, используя следующую теорему.

Теорема S. *Пусть L_i и p_i — числа, определенные выше. Размещение записей в таблице оптимально тогда и только тогда, когда*

$$p_1/L_1 \geq p_2/L_2 \geq \cdots \geq p_N/L_N. \tag{20}$$

Другими словами, минимальное значение

$$p_{a_1}L_{a_1} + p_{a_2}(L_{a_1} + L_{a_2}) + \cdots + p_{a_N}(L_{a_1} + \cdots + L_{a_N})$$

по всем перестановкам $a_1 a_2 \ldots a_N$ из множества $\{1, 2, \ldots, N\}$ равно (19) тогда и только тогда, когда выполняется условие (20).

Доказательство. Предположим, что мы поменяли местами на ленте R_i и R_{i+1}. Тогда значение величины (19) станет равным не

$$\cdots + p_i(L_1 + \cdots + L_{i-1} + L_i) + p_{i+1}(L_1 + \cdots + L_{i+1}) + \cdots,$$

а

$$\cdots + p_{i+1}(L_1 + \cdots + L_{i-1} + L_{i+1}) + p_i(L_1 + \cdots + L_{i+1}) + \cdots.$$

При этом изменение будет равно $p_iL_{i+1} - p_{i+1}L_i$. Если предположить оптимальность размещения (19), то любая перемена мест двух соседних записей должна приводить к увеличению времени работы, т. е. $p_i/L_i > p_{i+1}/L_{i+1}$. Таким образом, поскольку из оптимальности размещения следует набор неравенств (20), нами доказана необходимость условия (20) для оптимального размещения.

Докажем теперь достаточность выполнения условия (20) для оптимальности размещения. Приведенные выше рассуждения доказывают "локальную оптимальность" расположения — в том смысле, что любая перестановка двух рядом стоящих записей приведет к увеличению среднего времени работы. Однако это не доказывает невозможности сложного многоступенчатого обмена для улучшения производительности поиска, не доказывает, так сказать, "глобальной оптимальности". Мы рассмотрим два доказательства, в одном из которых используются знания из области компьютерных наук, а другое основано на некоторой математической хитрости.

Первое доказательство. Предположим, что условие (20) выполнено. Известно, что любую перестановку записей можно привести к "рассортированному" состоянию, т. е. привести ее к виду $R_1 R_2 \ldots R_N$ с использованием только перестановок двух соседних элементов. Каждая из таких перестановок заменяет $\ldots R_j R_i \ldots$ на $\ldots R_i R_j \ldots$ для некоторых $i < j$, тем самым уменьшая время поиска на неотрицательную величину $p_i L_j - p_j L_i$. Следовательно, порядок расположения записей $R_1 R_2 \ldots R_N$ должен иметь минимальное время поиска, т. е. быть оптимальным.

Второе доказательство. Заменим каждую вероятность p_i на

$$p_i(\epsilon) = p_i + \epsilon^i - (\epsilon^1 + \epsilon^2 + \cdots + \epsilon^N)/N, \tag{21}$$

где ϵ — очень малое положительное число. При этом равенство $x_1 p_1(\epsilon) + \cdots + x_N p_N(\epsilon) = y_1 p_1(\epsilon) + \cdots + y_N p_N(\epsilon)$ справедливо только в том случае, когда $x_1 = y_1$, $\ldots$, $x_N = y_N$. Отсюда, в частности, следует, что в (20) равенство никогда не будет выполняться. Рассмотрим $N!$ перестановок записей. Среди них есть, по меньшей мере, одна оптимальная, которая в соответствии с первой частью доказательства удовлетворяет условию (20). Однако поскольку теперь в условии (20) равенства невозможны, то и оптимальная перестановка может быть только одна. Следовательно, условие (20) однозначно определяет некоторую оптимальную перестановку для вероятностей $p_i(\epsilon)$, причем ϵ достаточно мало. Исходя из непрерывности тот же порядок должен быть оптимален и при $\epsilon = 0$. (Такой тип доказательств нередко используется в комбинаторной оптимизации.)

Теорема S была доказана В. И. Смитом (W. E. Smith) [*Naval Research Logistics Quarterly* **3** (1956), 59–66]. В приведенных ниже упражнениях содержатся дополнительные результаты оптимальной организации файлов.

УПРАЖНЕНИЯ

1. [*M20*] Пусть все ключи поиска равновероятны. Определите стандартное среднеквадратичное отклонение числа сравнений при успешном последовательном поиске в таблице с N записями.

2. [*15*] Измените алгоритм S для использования связанных записей вместо индексов. (Если P указывает на запись в таблице, полагаем, что KEY(P) — ключ, INFO(P) — связанная с ключом информация и LINK(P) — указатель на следующую запись. Полагаем также, что FIRST указывает на первую запись, а последняя запись указывает на Λ.)

3. [*16*] Напишите MIX-программу для алгоритма из упр. 2. Чему равно время выполнения программы (с использованием C и S из (1))?

▶ **4.** [*17*] Можно ли использовать идею алгоритма Q для таблиц в виде связанных записей (см. упр. 2)?

5. [*20*] Программа Q′ выполняется существенно быстрее программы Q при больших значениях C. Существуют ли значения C и S, при которых программа Q′ будет выполняться дольше, чем программа Q?

▶ **6.** [*20*] Добавьте три инструкции в программу Q′, которые позволят ей выполняться за время около $(3.33C + \text{constant})u$.

7. [*M20*] Определите среднее число сравнений (3) в случае "бинарного" распределения вероятности (5).

8. [*HM22*] Найдите асимптотический ряд для $H_n^{(x)}$ при $n \to \infty$; $x \neq 1$.

▶ **9.** [*HM28*] В тексте отмечено, что распределения вероятностей, данные в (11), (13) и (16), приблизительно одинаковы при $0 < \theta < 1$ и что среднее число сравнений с использованием (13) равно $\frac{\theta}{\theta+1}N + O(N^{1-\theta})$.

a) Означает ли это, что число сравнений равно $\frac{\theta}{\theta+1}N + O(N^{1-\theta})$ при использовании распределения (11)?

b) Верно ли это утверждение для распределения (16)?

c) Сравните (11) и (16) с (13) при $\theta < 0$.

10. [*M20*] Наилучшее расположение записей в последовательной таблице определяется условием (4). А что собой представляет *наихудшее* расположение? Покажите, что имеется простое соотношение между средним количеством сравнений при наилучшем и наихудшем размещениях записей.

11. [*M30*] Цель этого упражнения заключается в анализе предельного поведения самоорганизующегося файла при использовании эвристического метода перемещения записи в начало файла. Сначала введем некоторые обозначения. Пусть $f_m(x_1, x_2, \ldots, x_m)$ равно бесконечной сумме всех различных упорядоченных произведений $x_{i_1} x_{i_2} \ldots x_{i_k}$, таких, что $1 \le i_1, \ldots, i_k \le m$, причем каждое $x_1, x_2, \ldots, x_m$ входит во все произведения. Например,

$$f_2(x,y) = \sum_{j,k\ge 0} (x^{1+j}y(x+y)^k + y^{1+j}x(x+y)^k) = \frac{xy}{1-x-y}\left(\frac{1}{1-x} + \frac{1}{1-y}\right).$$

Исходя из множества X из n переменных $\{x_1, \ldots, x_n\}$, положим

$$P_{nm} = \sum_{1\le j_1<\cdots<j_m\le n} f_m(x_{j_1}, \ldots, x_{j_m}); \qquad Q_{nm} = \sum_{1\le j_1<\cdots<j_m\le n} \frac{1}{1-x_{j_1}-\cdots-x_{j_m}}.$$

Например, $P_{32} = f_2(x_1,x_2) + f_2(x_1,x_3) + f_2(x_2,x_3)$ и $Q_{32} = 1/(1-x_1-x_2) + 1/(1-x_1-x_3) + 1/(1-x_2-x_3)$. По определению полагаем $P_{n0} = Q_{n0} = 1$.

a) Предположим, что в самоорганизующийся файл запросы на поиск элемента R_i поступают с вероятностью p_i. Покажите, что после достаточно длительной работы системы элемент R_i оказывается на m-м месте с предельной вероятностью $p_i P_{(N-1)(m-1)}$, где множество X представляет собой $\{p_1, \ldots, p_{i-1}, p_{i+1}, \ldots, p_N\}$.

b) Суммируя результат (a) для $m = 1, 2, \ldots$, получаем тождество

$$P_{nn} + P_{n(n-1)} + \cdots + P_{n0} = Q_{nn}.$$

Получите отсюда следующие соотношения:

$$P_{nm} + \binom{n-m+1}{1}P_{n(m-1)} + \cdots + \binom{n-m+m}{m}P_{n0} = Q_{nm};$$

$$Q_{nm} - \binom{n-m+1}{1}Q_{n(m-1)} + \cdots + (-1)^m\binom{n-m+m}{m}Q_{n0} = P_{nm}.$$

c) Вычислите предельное среднее расстояние $d_i = \sum_{m\ge 1} m p_i P_{N-1,m-1}$ записи R_i от начала таблицы; затем вычислите $\widetilde{C}_N = \sum_{i=1}^N p_i d_i$.

12. [*M23*] Используйте (17) для вычисления среднего количества сравнений, необходимого для поиска в самоорганизующемся файле при бинарном распределении вероятностей ключей поиска (5).

13. [*M27*] Используя (17), вычислите $\widetilde{C}_N$ для распределения вероятностей (6).

14. [*M21*] Даны две последовательности действительных чисел — $\langle x_1, x_2, \ldots, x_n\rangle$ и $\langle y_1, y_2, \ldots, y_n\rangle$. Какая перестановка индексов $a_1 a_2 \ldots a_n$ делает сумму $\sum_i x_i y_{a_i}$ максимальной, минимальной?

▶ **15.** [*M22*] В тексте было показано, как оптимально расположить программы на ленте системной библиотеки для поиска только одной программы. Однако при работе с библиотекой *подпрограмм*, когда для работы программы пользователя необходимо загрузить различные подпрограммы, следует принять другой набор предположений.

В этом случае предположим, что запрос на подпрограмму j поступает с вероятностью P_j, причем вызовы различных подпрограмм независимы. Тогда, например, вероятность того, что не потребуется ни одна подпрограмма, равна $(1-P_1)(1-P_2)\ldots(1-P_N)$, а вероятность того, что поиск прекратится после загрузки j-й подпрограммы, равна $P_j(1-P_{j+1})\ldots(1-P_N)$. Если L_j — длина j-й подпрограммы, то среднее время поиска будет пропорционально

$$L_1P_1(1-P_2)\ldots(1-P_N)+(L_1+L_2)P_2(1-P_3)\ldots(1-P_N)+\cdots+(L_1+\cdots+L_N)P_N.$$

Каким в этом случае должно быть оптимальное расположение подпрограмм на ленте?

16. [*M22*] (Г. Ризель (H. Riesel).) Зачастую необходимо проверить, выполняются ли одновременно n заданных условий. (Например, может понадобиться проверить, что $x>0$ и $y<z^2$, и при этом не ясно, какое условие должно проверяться первым.) Предположим, что проверка j-го условия занимает T_j единиц времени и что условие выполняется с вероятностью p_j (причем эта вероятность не зависит от результатов выполнения других условий). В каком порядке должны выполняться проверки?

17. [*M23*] (В. И. Смит (W. E. Smith).) Предположим, имеется n заданий; j-е задание занимает T_j единиц времени; крайний срок его выполнения — D_j. Другими словами, j-е задание должно быть выполнено не позже момента D_j. Какое расписание работ $a_1\,a_2\ldots a_n$ минимизирует *максимальное запаздывание*, т е.

$$\max(T_{a_1}-D_{a_1},\,T_{a_1}+T_{a_2}-D_{a_2},\,\ldots,\,T_{a_1}+T_{a_2}+\cdots+T_{a_n}-D_{a_n})\,?$$

18. [*M30*] (*Сцепленный поиск.*) Предположим, что N записей расположены в памяти в виде линейного массива $R_1\ldots R_N$ и вероятность поиска записи R_j равна p_j. Поиск называется "сцепленным", если каждый последующий поиск начинается с того места, где завершился предыдущий. Если последовательные поиски независимы, то среднее время поиска составляет $\sum\limits_{1\le i,j\le N} p_ip_jd(i,j)$, где $d(i,j)$ — время, которое необходимо для поиска, начинающегося в позиции i и заканчивающегося в позиции j. Эта модель может быть применима, например, ко времени поиска дискового файла (при этом $d(i,j)$ представляет собой время перемещения между i- и j-м цилиндрами диска).

Цель данного упражнения — описать оптимальное размещение записей для сцепленного поиска в случае, когда $d(i,j)$ представляет собой монотонно возрастающую функцию от $|i-j|$, т.е. $d(i,j)=d_{|i-j|}$ и $d_1<d_2<\cdots<d_{N-1}$ (величина d_0 не существенна). Докажите, что размещение будет оптимально тогда и только тогда, когда будет выполняться условие либо $p_1\le p_N\le p_2\le p_{N-1}\le\cdots\le p_{\lfloor N/2\rfloor+1}$, либо $p_N\le p_1\le p_{N-1}\le p_2\le\cdots\le p_{\lceil N/2\rceil}$. (Другими словами, наилучшее расположение — в виде "органных труб", показанных на рис. 2.) *Указание.* Рассмотрите расположение, которому соответствуют вероятности $q_1\,q_2\ldots q_k\,s\,r_k\ldots r_2\,r_1\,t_1\ldots t_m$ для некоторых $m\ge 0$ и $k>0$; $N=2k+m+1$. Покажите, что расположение $q'_1\,q'_2\ldots q'_k\,s\,r'_k\ldots r'_2\,r'_1\,t_1\ldots t_m$ лучше, если $q'_i=\min(q_i,r_i)$ и $r'_i=\max(q_i,r_i)$, за исключением случая, когда $q'_i=q_i$ и $r'_i=r_i$ для всех i, и случая $q'_i=r_i$, $r'_i=q_i$ и $t_j=0$ для всех i и j. То же самое справедливо при отсутствии s и $N=2k+m$.

19. [*M20*] Продолжая выполнять упр. 18, найдите оптимальное расположение для сцепленного поиска, если функция $d(i,j)$ обладает следующим свойством: $d(i,j)+d(j,i)=c$ для всех $i\ne j$. (Такая ситуация возможна, например, при поиске на ленте без возможности обратной перемотки и с неизвестным нам направлением поиска; для $i<j$ имеем $d(i,j)=$

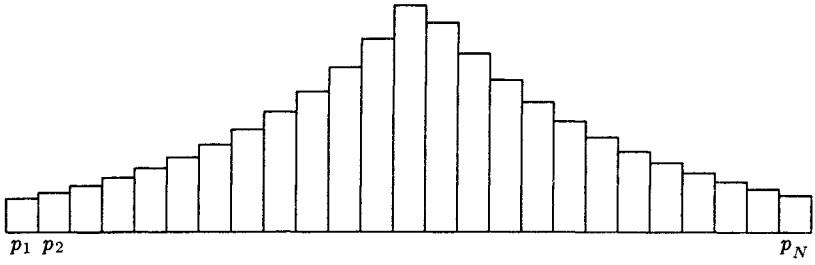


Рис. 2. Расположение в виде "органных труб" минимизирует среднее время сцепленного поиска.

$a + b(L_{i+1} + \cdots + L_j)$ и $d(j,i) = a + b(L_{j+1} + \cdots + L_N) + r + b(L_1 + \cdots + L_i)$, где r — время перемотки.)

20. [*M28*] Продолжая выполнять упр. 18, найдите оптимальное расположение для сцепленного поиска в случае, когда $d(i,j) = \min(d_{|i-j|}, d_{n-|i-j|})$ и $d_1 < d_2 < \cdots$. (Эта ситуация встречается, например, в двусвязном циклическом списке или в запоминающем устройстве с возможностью перемещения в обе стороны.)

21. [*M28*] Рассмотрим n-мерный куб с координатами вершин $(d_1, \ldots, d_n)$, где $d_j = 0$ или 1; две вершины называются *соседними*, если они различаются только одной координатой. Предположим, что набор из 2^n чисел $x_0 \le x_1 \le \cdots \le x_{2^n-1}$ должен быть сопоставлен 2^n вершинам таким образом, чтобы минимизировать сумму $\sum_{i,j} |x_i - x_j|$; сумма берется по всем i и j, таким, что x_i и x_j сопоставлены соседним вершинам. Докажите, что этот минимум достигается тогда, когда для всех j x_j сопоставлено вершине, координаты которой являются двоичным представлением числа j.

▶ **22.** [*20*] Предположим, что в большом файле вам необходимо найти 1 000 *ближайших* к данному ключу записей, т. е. записей, для которых функция расстояния $d(K_j, K)$ принимает наименьшие значения. Какая структура данных будет самой подходящей для такого последовательного поиска?

Пытайся до конца, сомнений прочь оскал,
И, все преодолев, найдешь ты, что искал.*

— РОБЕРТ ХЕРРИК (ROBERT HERRICK),
Ищите и обрящете (Seeke and finde) (1648)

* Перевод Светланы Тригуб.

6.2. ПОИСК ПУТЕМ СРАВНЕНИЯ КЛЮЧЕЙ

В этом разделе мы обсудим методы поиска, основанные на линейном упорядочении ключей (таком, как числовое или алфавитное). После сравнения данного аргумента K с ключом K_i из таблицы поиск продолжается одним из трех путей, в зависимости от того, какое из условий — $K < K_i$, $K = K_i$ или $K > K_i$ — истинно. Последовательные методы поиска, рассмотренные в разделе 6.1, по сути, имели только два варианта ветвления поиска — в зависимости от выполнения или не выполнения условия $K = K_i$; при отказе от исключительно последовательного доступа к таблице можно организовать более эффективный поиск с использованием отношения порядка.

6.2.1. Поиск в упорядоченной таблице

Что вы станете делать, если вам вручат большой телефонный справочник и попросят найти владельца телефона 444-3522? Вряд ли вы сможете предложить что-то лучшее, чем метод последовательного поиска из раздела 6.1 (методы наподобие "позвонить и спросить, кто это" и "купить компакт-диск с телефонной базой данных службы '09'" не рассматриваются). Беда в том, что, хотя в справочнике и содержится вся необходимая информация для поиска как по имени абонента, так и по его номеру, поиск по имени владельца гораздо проще именно благодаря упорядочению записей в телефонном справочнике. Последовательный поиск в огромном файле — это почти всегда безумие и пустая трата времени, но если файл упорядочен, решить задачу намного проще.

Вопросы сортировки рассматривались в главе 5, а потому нам не сложно выбрать подходящий метод сортировки, упорядочить данные и тем самым существенно облегчить задачу поиска. Само собой, выполнить однократный поиск в некоторой таблице проще последовательным методом, безо всякой сортировки; однако, если необходим многократный поиск, затраты на сортировку окупятся очень быстро. Таким образом, в этом разделе мы рассмотрим методы, обеспечивающие эффективный поиск в таблице, в которой ключи удовлетворяют условию

$$K_1 < K_2 < \cdots < K_N$$

и в которой мы имеем простой и быстрый доступ к ключу в любой заданной позиции. После сравнения в такой таблице K и K_i мы получим один из трех результатов:

- $K < K_i$ [$R_i, R_{i+1}, \ldots, R_N$ исключаются из рассмотрения];
- $K = K_i$ [поиск завершен];
- $K > K_i$ [$R_1, R_2, \ldots, R_i$ исключаются из рассмотрения].

В каждом из них в процессе поиска достигается существенный прогресс — за исключением тех случаев, когда i близкó к одному из концов таблицы (более корректно было бы сказать "к одному из концов рассматриваемой части таблицы". — *Прим. перев.*). Таким образом, с помощью упорядочения можно построить эффективные алгоритмы поиска.

Бинарный поиск. Вероятно, первым методом, который приходит в голову, является следующий: сравнить K со средним ключом в таблице, в результате этого сравнения определить, в какой половине таблицы находится искомый ключ, и снова

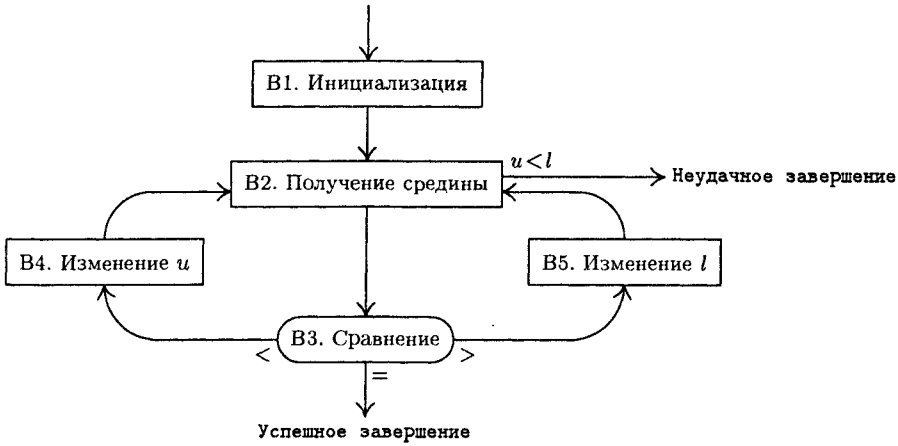


Рис. 3. Бинарный поиск.

применить ту же процедуру к половине таблицы. Максимум за величину порядка $\lg N$ сравнений мы найдем искомый ключ (либо установим, что его нет в таблице). Эта процедура известна как "логарифмический поиск" или "метод деления пополам" но чаще всего употребляется термин *бинарный поиск*.

Хотя основная идея бинарного поиска сравнительно проста, детали его реализации нетривиальны и много неплохих программистов ошибались при первых попытках написать соответствующую программу. В одной из наиболее популярных форм реализации метода используются два указателя, l и u, которые указывают верхнюю и нижнюю границы поиска.

Алгоритм B (*Бинарный поиск (Binary search)*). Дана таблица записей $R_1 R_2 \ldots R_N$, ключи которых расположены в порядке возрастания: $K_1 < K_2 < \cdots < K_N$; алгоритм используется для поиска в таблице заданного аргумента K.

B1. [Инициализация.] Установить $l \leftarrow 1$, $u \leftarrow N$.

B2. [Получение средины.] (На этом шаге мы знаем, что если K имеется в таблице, то справедливо условие $K_l \le K \le K_u$. Более точное описание ситуации можно найти в приведенном ниже упр. 1.) Если $u < l$, алгоритм завершается неудачно; в противном случае следует установить $i \leftarrow \lfloor (l+u)/2 \rfloor$, чтобы i соответствовало примерно средине рассматриваемой части таблицы.

B3. [Сравнение.] Если $K < K_i$, перейти к шагу B4; если $K > K_i$, перейти к шагу B5; если $K = K_i$, алгоритм успешно завершается.

B4. [Изменение u.] Установить $u \leftarrow i - 1$ и перейти к шагу B2.

B5. [Изменение l.] Установить $l \leftarrow i + 1$ и перейти к шагу B2. ▮

На рис. 3 приведена блок-схема алгоритма, а на рис. 4 показаны два случая проведения бинарного поиска: в первом разыскивается число 653, имеющееся в таблице, а во втором — число 400, отсутствующее в таблице. Скобки показывают положение указателей l и u, а подчеркнутое число — K_i. В обоих случаях поиск завершается после выполнения четырех сравнений.

a) Поиск 653:

```
[061 087 154 170 275 426 503 509 512 612 653 677 703 765 897 908]
 061 087 154 170 275 426 503 509 [512 612 653 677 703 765 897 908]
 061 087 154 170 275 426 503 509 [512 612 653] 677 703 765 897 908
 061 087 154 170 275 426 503 509 512 612 [653] 677 703 765 897 908
```

b) Поиск 400:

```
[061 087 154 170 275 426 503 509 512 612 653 677 703 765 897 908]
[061 087 154 170 275 426 503] 509 512 612 653 677 703 765 897 908
 061 087 154 170 [275 426 503] 509 512 612 653 677 703 765 897 908
 061 087 154 170 [275] 426 503 509 512 612 653 677 703 765 897 908
 061 087 154 170 275] [426 503 509 512 612 653 677 703 765 897 908
```

Рис. 4. Примеры бинарного поиска.

Программа B (*Бинарный поиск*). Как и в программах из раздела 6.1, полагаем, что K_i представляет собой полное слово, находящееся по адресу KEY $+ i$. В приведенном коде rI1 $\equiv l$, rI2 $\equiv u$, rI3 $\equiv i$.

01	START	ENT1	1	1	*B1. Инициализация.* $l \leftarrow 1$.
02		ENT2	N	1	$u \leftarrow N$.
03		JMP	2F	1	Переход к шагу B2.
04	5H	JE	SUCCESS	$C1$	Переход, если $K = K_i$.
05		ENT1	1,3	$C1 - S$	*B5. Изменение l.* $l \leftarrow i + 1$.
06	2H	ENTA	0,1	$C + 1 - S$	*B2. Получение средины.*
07		INCA	0,2	$C + 1 - S$	rA $\leftarrow l + u$.
08		SRB	1	$C + 1 - S$	rA $\leftarrow \lfloor$rA$/2\rfloor$. (Изменяется также rX.)
09		STA	TEMP	$C + 1 - S$	
10		CMP1	TEMP	$C + 1 - S$	
11		JG	FAILURE	$C + 1 - S$	Переход, если $u < l$.
12		LD3	TEMP	C	$i \leftarrow$ midpoint.
13	3H	LDA	K	C	*B3. Сравнение.*
14		CMPA	KEY,3	C	
15		JGE	5B	C	Переход, если $K \geq K_i$.
16		ENT2	-1,3	$C2$	*B4. Изменение u.* $u \leftarrow i - 1$.
17		JMP	2B	$C2$	Переход к шагу B2. ▮

Эта процедура не вполне удачно реализуется на компьютере MIX в связи с небогатой арифметикой индексных регистров. Время работы составляет $(18C - 10S + 12)u$, где $C = C1 + C2$ — количество произведенных сравнений (количество выполнений шага B3) и $S = 1$ при успешном (или 0 при неудачном) исходе поиска. Обратите внимание, что операция в строке 08 — SRB (shift right binary 1 — побитовый сдвиг вправо на единицу) — допустима лишь в бинарных версиях MIX; в общем случае ее следует изменить на MUL =1//2+1=, что приведет к увеличению времени работы программы до $(26C - 18S + 20)u$.

Представление в виде дерева. Для лучшего понимания работы алгоритма B представим описанную процедуру в виде бинарного дерева принятия решения, как показано на рис. 5, при $N = 16$.

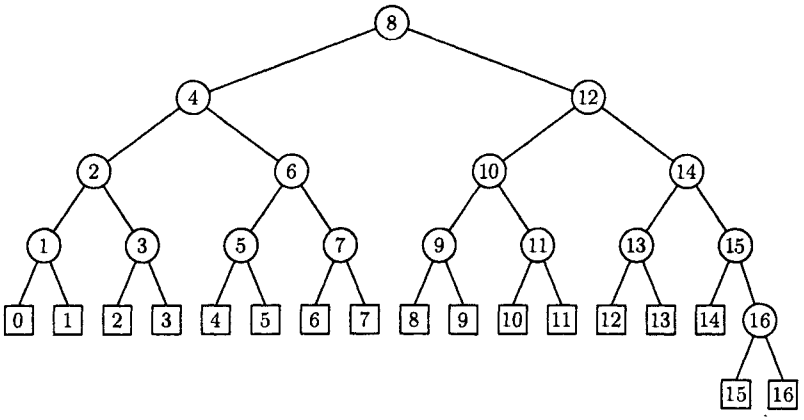


Рис. 5. Дерево сравнений, соответствующее бинарному поиску для $N = 16$.

При $N = 16$ сначала согласно алгоритму сравниваются K и K_8; на рисунке это показано корневым узлом (8). Затем, если $K < K_8$, алгоритм переходит в левое поддерево и сравниваются K с K_4; точно так же при $K > K_8$ будет выполняться работа с правым поддеревом. Неудачный поиск приведет в один из внешних "квадратных" узлов, пронумерованных от $\boxed{0}$ до $\boxed{N}$, например узел $\boxed{6}$ будет достигнут нами только при $K_6 < K < K_7$.

Бинарное дерево, соответствующее бинарному поиску по N записям, может быть построено следующим образом: если $N = 0$, дерево представляет собой просто один узел $\boxed{0}$. В противном случае корневой узел —

$$\left(\lceil N/2 \rceil\right).$$

При этом левое поддерево представляет собой бинарное дерево с $\lceil N/2 \rceil - 1$ узлами, а правое — с $\lfloor N/2 \rfloor$ узлами (все числа в узлах увеличены на $\lceil N/2 \rceil$).

Точно так в виде бинарного дерева с N узлами, в котором все узлы пронумерованы от 1 до N, может быть представлен *любой* алгоритм поиска в упорядоченной таблице длиной N (если только алгоритм не выполняет излишние сравнения). Верно и обратное — любое бинарное дерево соответствует некоторому методу поиска в упорядоченной таблице; достаточно просто пометить узлы

$$\boxed{0} \quad (1) \quad \boxed{1} \quad (2) \quad \boxed{2} \quad \ldots \quad \boxed{N-1} \quad (N) \quad \boxed{N} \tag{1}$$

в симметричном порядке слева направо.

Если аргумент поиска алгоритма B равен K_{10}, алгоритм выполняет сравнения $K > K_8$, $K < K_{12}$, $K = K_{10}$. На рис. 5 это соответствует пути от корня дерева к вершине (10). Точно так поведение алгоритма B при поиске других ключей соответствует некоторому пути из корня дерева. Метод построения бинарных деревьев, соответствующих алгоритму B, позволяет легко доказать с помощью индукции по N следующую теорему.

Теорема B. *Для $2^{k-1} \le N < 2^k$ успешный поиск по алгоритму B требует* (min 1, max k) *сравнений; неудачный поиск при $2^{k-1} \le N < 2^k - 1$ требует $k - 1$ либо k сравнений.* ▌

Дальнейший анализ бинарного поиска. (Читателям, для которых математика не представляет интереса, рекомендуем пропустить материал до формулы (4).) Представление в виде дерева показывает нам также простой путь вычисления *среднего* числа сравнений. Пусть C_N — среднее число сравнений при успешном поиске; предположим также, что все N ключей равновероятны в качестве аргумента поиска. Среднее число сравнений при *не*удачном поиске — C'_N; все $N+1$ интервалов внутри и вне граничных значений также равновероятны. Тогда по определению длин внутреннего и внешнего путей

$$C_N = 1 + \frac{\text{длина внутреннего пути дерева}}{N}, \qquad C'_N = \frac{\text{длина внешнего пути дерева}}{N+1}.$$

Из 2.3.4.5–(3) видно, что длина внешнего пути всегда на $2N$ больше длины внутреннего пути. Отсюда следует несколько неожиданное соотношение между C_N и C'_N:

$$C_N = \left(1 + \frac{1}{N}\right) C'_N - 1. \tag{2}$$

Эта формула, выведенная Т. Н. Хиббардом (T. N. Hibbard) [*JACM* **9** (1962), 16–17], справедлива для всех методов, соответствующих бинарным деревьям; другими словами, она справедлива для всех методов, не использующих излишних сравнений. Дисперсия среднего числа сравнений при успешном поиске также может быть выражена через дисперсию среднего числа сравнений при неудачном поиске (см. упр. 25).

На основании приведенных выше формул мы можем сказать, что "наилучший" путь поиска методом сравнения тот, дерево которого имеет минимальную длину внешнего пути среди всех бинарных деревьев с N внутренними узлами. К счастью, можно доказать, что в этом смысле *алгоритм B оптимален* для любого N (как помните, в упр. 5.3.1–20 было доказано, что бинарное дерево имеет минимальную длину пути тогда и только тогда, когда все внешние узлы расположены не более чем на двух соседних уровнях). Отсюда следует, что длина внешнего пути дерева, соответствующего алгоритму B, равна

$$(N+1)(\lfloor \lg N \rfloor + 2) - 2^{\lfloor \lg N \rfloor + 1}. \tag{3}$$

(См. 5.3.1–(34).) Из формул (2) и (3) мы можем вычислить точные средние значения количества сравнений, полагая, что все аргументы поиска равновероятны.

$N=$	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
$C_N=$	1	$1\frac{1}{2}$	$1\frac{2}{3}$	2	$2\frac{1}{5}$	$2\frac{2}{6}$	$2\frac{3}{7}$	$2\frac{5}{8}$	$2\frac{7}{9}$	$2\frac{9}{10}$	3	$3\frac{1}{12}$	$3\frac{2}{13}$	$3\frac{3}{14}$	$3\frac{4}{15}$	$3\frac{6}{16}$
$C'_N=$	1	$1\frac{2}{3}$	2	$2\frac{2}{5}$	$2\frac{4}{6}$	$2\frac{6}{7}$	3	$3\frac{2}{9}$	$3\frac{4}{10}$	$3\frac{6}{11}$	$3\frac{8}{12}$	$3\frac{10}{13}$	$3\frac{12}{14}$	$3\frac{14}{15}$	4	$4\frac{2}{17}$

В общем случае при $k = \lfloor \lg N \rfloor$ мы имеем следующее:

$$\begin{aligned} C_N &= k + 1 - (2^{k+1} - k - 2)/N &&= \lg N - 1 + \epsilon + (k+2)/N, \\ C'_N &= k + 2 - 2^{k+1}/(N+1) &&= \lg(N+1) + \epsilon', \end{aligned} \tag{4}$$

где $0 \le \epsilon, \epsilon' < 0.0861$ (см. 5.3.1–(35)).

Итак, алгоритм B никогда не делает более $\lfloor \lg N \rfloor + 1$ сравнений; среднее количество сравнений при успешном поиске составляет около $\lg N - 1$. Никакой другой

метод поиска, основанный на сравнении ключей, не может превзойти этот результат. Среднее время работы программы B составляет приблизительно

$$\begin{aligned} &(18\lg N - 16)u \qquad \text{для успешного поиска,} \\ &(18\lg N + 12)u \qquad \text{для неудачного поиска} \end{aligned} \tag{5}$$

(в предположении равновероятных исходов поиска).

Важное изменение. Вместо трех указателей (l, i и u) в процессе поиска можно использовать только два: текущее положение i и величину его изменения δ. После каждого сравнения, не давшего равенства, мы можем установить $i \leftarrow i \pm \delta$ и $\delta \leftarrow \delta/2$ (приблизительно). Это вполне возможный путь, хотя и требующий исключительного внимания к мельчайшим деталям, как в приведенном ниже алгоритме. Более простые подходы будут неработоспособны.

Алгоритм U (*Однородный бинарный поиск* (*Uniform binary search*)). Дана таблица записей $R_1, R_2, \ldots, R_N$, ключи которых расположены в порядке возрастания: $K_1 < K_2 < \cdots < K_N$. Алгоритм обеспечивает поиск заданного аргумента K. В случае четного N алгоритм будет обращаться к фиктивному ключу K_0, значение которого должно быть равно $-\infty$ (или любому значению, меньшему K). Алгоритм работает в предположении, что $N \geq 1$.

U1. [Инициализация.] Установить $i \leftarrow \lceil N/2 \rceil$, $m \leftarrow \lfloor N/2 \rfloor$.

U2. [Сравнение.] Если $K < K_i$, перейти к шагу U3; если $K > K_i$, перейти к шагу U4; если $K = K_i$, алгоритм успешно завершен.

U3. [Уменьшить i.] (Мы определили интервал продолжения поиска, содержащий m или $m - 1$ записей; i указывает на первый элемент справа от интервала.) При $m = 0$ алгоритм завершается неудачно. В противном случае следует сначала установить $i \leftarrow i - \lceil m/2 \rceil$, а затем — $m \leftarrow \lfloor m/2 \rfloor$ и перейти к шагу U2.

U4. [Увеличение i.] (Мы определили интервал продолжения поиска, содержащий m или $m - 1$ записей; i указывает на первый элемент слева от интервала.) При $m = 0$ алгоритм завершается неудачно. В противном случае следует сначала установить $i \leftarrow i + \lceil m/2 \rceil$, а затем — $m \leftarrow \lfloor m/2 \rfloor$ и перейти к шагу U2. ▌

На рис. 6 показано бинарное дерево, соответствующее поиску при $N = 10$. В случае неудачного поиска алгоритм может выполнить излишнее сравнение непосредственно перед окончанием работы; такие узлы на рисунке заштрихованы. Мы называем этот процесс поиска *однородным* потому, что разность между числами узла на уровне l и его узла-предшественника на уровне $l - 1$ представляет собой константу δ для всех узлов на уровне l.

Теория, лежащая в основе алгоритма U, может быть пояснена следующим образом. Предположим, что у нас имеется интервал для поиска длиной $n - 1$; сравнение со средним элементом (для четного n) или с одним из средних элементов (для нечетного n) дает нам два интервала — длиной $\lfloor n/2 \rfloor - 1$ и $\lceil n/2 \rceil - 1$. После повторения этого процесса k раз мы получим 2^k интервалов, наименьший из которых имеет длину $\lfloor n/2^k \rfloor - 1$, а наибольший — $\lceil n/2^k \rceil - 1$. Следовательно, длина двух интервалов на одном уровне отличается не более чем на единицу; это делает возможным выбор "среднего" элемента без запоминания точных значений длин интервалов.

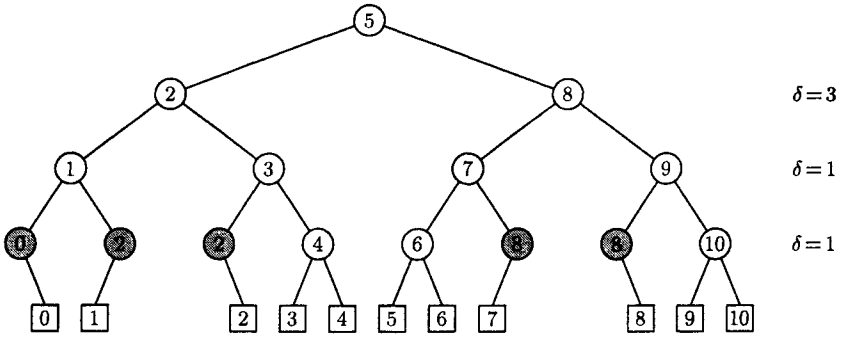


Рис. 6. Дерево сравнений для "однородного" бинарного поиска при $N = 10$.

Принципиальное достоинство алгоритма U заключается в том, что нам совершенно не нужно хранить значение m; необходима только ссылка на небольшую таблицу δ для использования на каждом уровне дерева. Таким образом, алгоритм сводится к следующей процедуре, одинаково подходящей как для бинарных, так и для десятичных компьютеров.

Алгоритм C (*Однородный бинарный поиск*). Этот алгоритм подобен алгоритму U, но использует вспомогательную таблицу вместо вычислений:

$$\texttt{DELTA}[j] = \left\lfloor \frac{N + 2^{j-1}}{2^j} \right\rfloor \qquad \text{для } 1 \le j \le \lfloor \lg N \rfloor + 2. \tag{6}$$

C1. [Инициализация.] Установить $i \leftarrow \texttt{DELTA}[1]$, $j \leftarrow 2$.

C2. [Сравнение.] Если $K < K_i$, перейти к шагу C3; если $K > K_i$, перейти к шагу C4; если $K = K_i$, алгоритм успешно завершается.

C3. [Уменьшение i.] Если $\texttt{DELTA}[j] = 0$, алгоритм завершается неудачно. В противном случае установить $i \leftarrow i - \texttt{DELTA}[j]$, $j \leftarrow j + 1$ и перейти к шагу C2.

C4. [Увеличение i.] Если $\texttt{DELTA}[j] = 0$, алгоритм завершается неудачно. В противном случае установить $i \leftarrow i + \texttt{DELTA}[j]$, $j \leftarrow j + 1$ и перейти к шагу C2. ▮

В упр. 8 доказывается, что настоящий алгоритм использует искусственный ключ $K_0 = -\infty$ только при четных N.

Программа C (*Однородный бинарный поиск*). Эта программа выполняет ту же задачу, что и программа B, используя алгоритм C; при этом rA $\equiv K$, rI1 $\equiv i$, rI2 $\equiv j$, rI3 $\equiv$ DELTA[j].

```
01  START   ENT1 N+1/2      1        C1. Инициализация. i ← ⌊(N+1)/2⌋.
02          ENT2 2          1        j ← 2.
03          LDA  K          1
04          JMP  2F         1
05  3H      JE   SUCCESS    C1       Переход, если K = K_i.
06          J3Z  FAILURE    C1 − S   Переход, если DELTA[j] = 0.
07          DEC1 0,3        C1 − S − A  C3. Уменьшение i.
08  5H      INC2 1          C − 1    j ← j + 1.
```

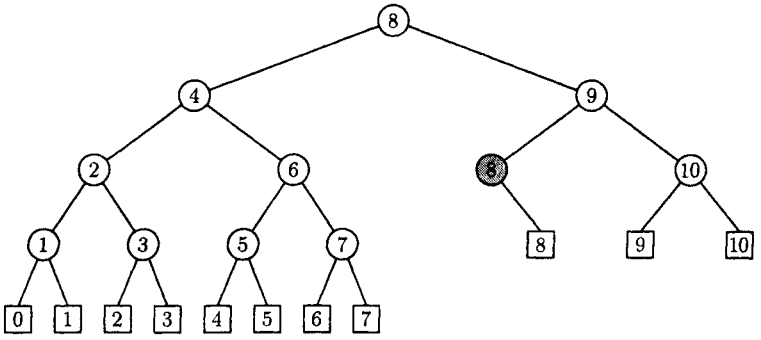


Рис. 7. Дерево сравнений для почти равномерного поиска по методу Шара при $N = 10$.

09	2H	LD3	DELTA,2	C	*C2. Сравнение.*
10		CMPA	KEY,1	C	
11		JLE	3B	C	Переход, если $K \le K_i$.
12		INC1	0,3	$C2$	*C4. Увеличение i.*
13		J3NZ	5B	$C2$	Переход, если DELTA[j] $\neq 0$.
14	FAILURE	EQU	*	$1 - S$	Выход при отсутствии в таблице. ▌

В случае успешного завершения поиска данный алгоритм соответствует бинарному дереву с такой же длиной внутреннего пути, как и в алгоритме B, поэтому среднее количество сравнений C_N остается прежним. При неудачном поиске алгоритм C всегда выполняет в точности $\lfloor \lg N \rfloor + 1$ сравнений. Общее время работы программы C не совсем симметрично по отношению к левым и правым ветвям, и C1 имеет больший вес, чем C2. Впрочем, в упр. 11 будет показано, что случаи $K < K_i$ имеют ту же вероятность, что и $K > K_i$. Следовательно, программа C работает приблизительно следующее время:

$$\begin{aligned} &(8.5 \lg N - 6)u && \text{при успешном поиске,} \\ &(8.5 \lfloor \lg N \rfloor + 12)u && \text{при неудачном поиске.} \end{aligned} \tag{7}$$

Это более чем в два раза быстрее программы B; при этом специфика бинарного компьютера не используется (в случае времени работы (5) программы B предполагалось наличие команды битового сдвига в компьютере MIX).

Другая модификация бинарного поиска, предложенная в 1971 году Л. Э. Шаром (L. E. Shar), на некоторых компьютерах будет работать еще быстрее, так как она однородна после первого шага и не требует использования таблицы. Первый шаг состоит в сравнении K и K_i, где $i = 2^k$, $k = \lfloor \lg N \rfloor$. Если $K < K_i$, мы используем однородный поиск с $\delta = 2^{k-1}, 2^{k-2}, \ldots, 1, 0$. С другой стороны, если $K > K_i$, мы устанавливаем $i = i' = N + 1 - 2^l$, где $l = \lceil \lg(N - 2^k + 1) \rceil$, и, делая вид, что первое сравнение на самом деле было $K > K_{i'}$, используем бинарный поиск с $\delta = 2^{l-1}, 2^{l-2}, \ldots, 1, 0$.

Метод Шара для $N = 10$ проиллюстрирован на рис. 7. Подобно предыдущим алгоритмам, он никогда не выполняет больше $\lfloor \lg N \rfloor + 1$ сравнений. Следовательно, выполняемое им количество сравнений превышает минимально возможное среднее значение не более чем на единицу — несмотря на то, что иногда алгоритму

приходится проводить несколько избыточных сравнений при успешном поиске (см. упр. 12).

Еще один вариант бинарного поиска, который осуществляется быстрее *всех описанных* при очень больших N, обсуждается в упр. 23. Но в упр. 24 вы найдете еще более быстрый метод...

***Поиск Фибоначчи.** При рассмотрении многофазного слияния мы видели, что числа Фибоначчи могут играть такую же роль, что и степени числа 2. Подобное явление наблюдается и при поиске, где числа Фибоначчи позволяют разработать альтернативу бинарному поиску. Для некоторых компьютеров предлагаемый метод предпочтителен в связи с тем, что в нем используются только сложение и вычитание (без деления на 2). Следует различать процедуру, которую мы начинаем обсуждать, и важный численный метод, также называемый "поиском Фибоначчи", используемый для поиска максимума унимодальной функции [см. *Fibonacci Quarterly* **4** (1966), 265–269]. Совпадение названий* нередко приводит к недоразумениям!

Технология поиска Фибоначчи, на первый взгляд, представляется весьма загадочной, и, если просто взять программу и постараться понять, как она работает, вам покажется, что это полное шаманство. Однако шаманство превратится в обычный танец с бубном, как только мы построим соответствующее дерево поиска. Поэтому наш рассказ начнется с *деревьев Фибоначчи*.

На рис. 8 показано дерево Фибоначчи порядка 6. Оно больше похоже на реальный, не очень хорошо подстриженный куст, чем на другие деревья, которые мы рассматривали. Возможно, это связано с тем, что многие природные процессы описываются законом Фибоначчи. В целом, дерево Фибоначчи порядка k имеет $F_{k+1} - 1$ внутренних (на рисунке — круглых) и F_{k+1} внешних (на рисунке — квадратных) узлов. Строится оно следующим образом.

Если $k = 0$ или $k = 1$, дерево вырождается в $\boxed{0}$.

Если $k \geq 2$, корнем является F_k; левое поддерево представляет собой дерево Фибоначчи порядка $k - 1$; правое поддерево представляет собой дерево Фибоначчи порядка $k - 2$ с числами, увеличенными на F_k.

Обратите внимание на то, что, за исключением внешних узлов, числа двух дочерних узлов каждого внутреннего узла отличаются от него на одну и ту же величину, и эта величина — не что иное, как число Фибоначчи (например, на рассматриваемом рисунке $5 = 8 - F_4$ и $11 = 8 + F_4$). Если разница на каком-либо уровне составляет F_j, то на следующем уровне она будет равна F_{j-1} для левой ветви и F_{j-2} — для правой. Так, например, $3 = 5 - F_3$, а $10 = 11 - F_2$.

Комбинируя эти наблюдения с механизмом распознавания внешних узлов, мы получим следующий алгоритм поиска.

Алгоритм F (*Поиск Фибоначчи* (*Fibonaccian search*)). Дана таблица записей $R_1 R_2 \ldots R_N$, ключи которых расположены в порядке возрастания $K_1 < K_2 < \cdots < K_N$. Алгоритм осуществляет поиск заданного аргумента K.

---

* Нужно отметить, что в английском языке такое совпадение превращается в обычную схожесть; метод поиска, который мы будем рассматривать, именуется "Fibonaccian search", а метод поиска максимума унимодальной функции — "Fibonacci search". — *Прим. перев.*

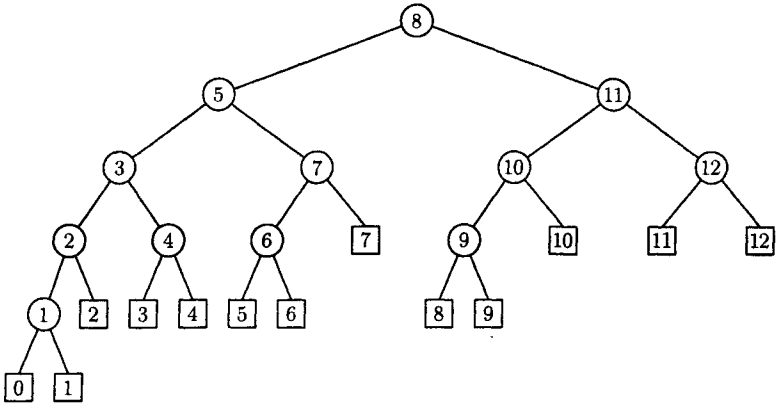


Рис. 8. Дерево Фибоначчи порядка 6.

Для удобства описания предполагается, что $N+1$ представляет собой число Фибоначчи F_{k+1} (выполнив соответствующую инициализацию, алгоритм несложно распространить на любые значения N; см. упр. 14).

F1. [Инициализация.] Установить $i \leftarrow F_k$, $p \leftarrow F_{k-1}$, $q \leftarrow F_{k-2}$. (В описании этого алгоритма p и q означают последовательные числа Фибоначчи.)

F2. [Сравнение.] Если $K < K_i$, перейти к шагу F3; если $K > K_i$, перейти к шагу F4; если $K = K_i$, алгоритм успешно завершается.

F3. [Уменьшение i.] Если $q = 0$, алгоритм завершается неудачно. В противном случае следует установить $i \leftarrow i - q$ и $(p, q) \leftarrow (q, p-q)$; затем перейти к шагу F2.

F4. [Увеличение i.] Если $p = 1$, алгоритм завершается неудачно. В противном случае следует сначала установить $i \leftarrow i + q$, $p \leftarrow p - q$, а затем — $q \leftarrow q - p$ и перейти к шагу F2. ▌

В приведенной ниже MIX-реализации алгоритма используется дублирование внутреннего цикла для повышения скорости работы. В одном из циклов p хранится в rI2, а q — в rI3; в другом цикле регистры меняются ролями. В действительности в регистрах хранятся значения $p-1$ и $q-1$, что облегчает проверку "$p = 1$?" на шаге F4.

Программа F (*Поиск Фибоначчи*). Мы придерживаемся предварительных соглашений; rA $\equiv K$, rI1 $\equiv i$, (rI2 или rI3) $\equiv p-1$, (rI3 или rI2) $\equiv q-1$.

01	START	LDA	K	1	*F1. Инициализация.*
02		ENT1	F_k	1	$i \leftarrow F_k$.
03		ENT2	F_{k-1}-1	1	$p \leftarrow F_{k-1}$.
04		ENT3	F_{k-2}-1	1	$q \leftarrow F_{k-2}$.
05		JMP	F2A	1	Переход к шагу F2.
06	F4A	INC1	1,3	$C2-S-A$	*F4. Увеличение i.* $i \leftarrow i+q$.
07		DEC2	1,3	$C2-S-A$	$p \leftarrow p-q$.
08		DEC3	1,2	$C2-S-A$	$q \leftarrow q-p$.
09	F2A	CMPA	KEY,1	C	*F2. Сравнение.*
10		JL	F3A	C	Переход к шагу F3, если $K < K_i$.
11		JE	SUCCESS	$C2$	Выход, если $K = K_i$.

```
12          J2NZ F4A        C2 − S     Переход к шагу F4, если p ≠ 1.
13          JMP  FAILURE    A          Выход при отсутствии в таблице.
14   F3A    DEC1 1,3        C1         F3. Уменьшение i. i ← i − q.
15          DEC2 1,3        C1         p ← p − q.
16          J3NN F2B        C1         Обмен регистрами при q > 0.
17          JMP  FAILURE    1 − S − A  Выход при отсутствии в таблице.
18   F4B    INC1 1,2                   (Строки 18–29 параллельны строкам 06–17.)
19          DEC3 1,2
20          DEC2 1,3
21   F2B    CMPA KEY,1
22          JL   F3B
23          JE   SUCCESS
24          J3NZ F4B
25          JMP  FAILURE
26   F3B    DEC1 1,2
27          DEC3 1,2
28          J2NN F2A
29          JMP  FAILURE                ▌
```

Время работы этой программы анализируется в упр. 18. Из рис. 8 видно (и анализ это доказывает), что левая ветвь выбирается чаще, чем правая. Обозначив количество выполнения шагов F2, F3 и F4 через C, $C1$ и $(C2 - S)$ соответственно, получим

$$\begin{aligned} C &= (\text{ave} \quad \phi k/\sqrt{5} + O(1), \quad \max\ k-1), \\ C1 &= (\text{ave} \quad k/\sqrt{5} + O(1), \quad \max\ k-1), \\ C2 - S &= (\text{ave}\ \phi^{-1} k/\sqrt{5} + O(1), \quad \max\ \lfloor k/2 \rfloor). \end{aligned} \tag{8}$$

Таким образом, левая ветвь выбирается примерно в $\phi \approx 1.618$ раз чаще, чем правая (вполне понятный и предсказуемый результат, поскольку после каждого сравнения оставшийся интервал делится на две части; левая часть при этом приблизительно в ϕ раз больше правой). В итоге суммарное среднее время работы программы F составляет около

$$\tfrac{1}{5}\big((18 + 4\phi)k + 31 - 26\phi\big)u \approx (7.050 \lg N + 1.08)u \tag{9}$$

для успешного поиска плюс $(9 - 3\phi)u \approx 4.15u$ — для неудачного. Это меньше, чем в случае использования программы C, хотя наихудшее время работы (около $8.6 \lg N$) несколько больше.

Интерполяционный поиск. Забудем ненадолго о компьютерах и посмотрим, как с задачей поиска справляется человек. Очень часто хороший алгоритм можно создать, проанализировав собственные действия в реальной жизни.

Представьте, что вы ищете слово в словаре. Вероятно, вы *не* станете открывать словарь посередине и идти к странице, расположенной на 1/4 или 3/4 от начала, т. е. выполнять алгоритм бинарного поиска. Шансы на то, что вы воспользуетесь поиском Фибоначчи, просто смехотворны. Но ведь вы находите нужное слово в словаре? Так как же вы это делаете?

Если слово начинается с буквы A, вероятно, вы ищете его в начале книги. Во многих словарях есть ярлычки, которые позволяют мгновенно найти страницу, на которой начинаются слова на заданную букву. Данная технология ускорения

поиска вполне применима в мире компьютеров; о таких алгоритмах мы поговорим в разделе 6.3.

Но и после этого первого шага ваши действия ничуть не похожи на то, что происходит при реализации какого-либо из рассмотренных алгоритмов. Если вы заметите, что искомое слово находится ближе к концу группы слов на нужную букву, вы просто пропустите изрядное количество страниц (или начнете поиск от следующей буквы к предыдущей). Вот в чем состоит основное отличие ваших действий от действий, выполняемых в алгоритмах, для которых понятия "немного больше" и "существенно больше" одинаковы.

Эти рассуждения приводят нас к алгоритму, который может быть назван *интерполяционным поиском*: если мы знаем, что K находится между K_l и K_u, следующую проверку мы будем делать примерно на расстоянии $(K - K_l)/(K_u - K_l)$ от l (в предположении, что ключи представляют собой числовые значения, близкие к арифметической прогрессии).

Асимптотически интерполяционный поиск превосходит по своим характеристикам бинарный поиск. В то время как один шаг бинарного поиска уменьшает количество проверяемых записей от n до $\frac{1}{2}n$, один шаг интерполяционного поиска уменьшает это количество до $\sqrt{n}$ (если ключи распределены в таблице случайным образом). В результате интерполяционный поиск требует в среднем $\lg\lg N$ шагов для уменьшения диапазона проверки от N до 2 (см. упр. 22).

Однако имитационные вычислительные эксперименты показали, что интерполяционный поиск не настолько снижает количество выполняемых сравнений, чтобы компенсировать требуемое для дополнительных вычислений время (пока таблица не *очень* велика). Кроме того, типичные таблицы недостаточно случайны, да и разница между значениями $\lg\lg N$ и $\lg N$ становится значительной только при больших N (скажем, при $N = 2^{16} = 65\,536$).

Интерполяция наиболее успешно применяется на ранних стадиях поиска в большом внешнем файле; после того как диапазон поиска существенно уменьшится, следует перейти к бинарному поиску. (В связи с этим заметим, что поиск в словаре представляет собой *внешний* поиск, о котором мы поговорим немного позже.)

История и библиография. Наиболее ранним известным примером рассортированного для упрощения поиска длинного списка элементов является известная вавилонская таблица обратных величин Инакибит-Ану, датируемая примерно 200 г. до н. э. Эта глиняная табличка содержит более 100 пар значений, которые представляют собой начало списка из приблизительно 500 многозначных шестидесятеричных чисел и их обратных величин, рассортированных в лексикографическом порядке. Например, список включает последовательность

01 13 09 34 29 08 08 53 20	49 12 27
01 13 14 31 52 30	49 09 07 12
01 13 43 40 48	48 49 41 15
01 13 48 40 30	48 46 22 59 25 25 55 33 20
01 14 04 26 40	48 36

Сортировка 500 подобных чисел при помощи технологий тех времен является просто феноменальной! [См. D. E. Knuth, *Selected Papers on Computer Science* (Cambridge Univ. Press, 1996), Chapter 11.]

Представляется совершенно естественной сортировка числовых значений, однако отношение порядка для букв или слов не столь очевидно. Тем не менее уже в наиболее древних алфавитах имелись последовательности букв, которые служили для их сравнения. Например, во многих библейских псалмах содержатся стихи, следующие друг за другом в строгом алфавитном порядке: первый стих начинается с альфы, второй — с беты и т. д., что облегчает запоминание. Часто стандартная последовательность букв использовалась семитами и греками для обозначения чисел*, например α, β и γ означали 1, 2 и 3 соответственно.

Алфавитный порядок слов, который представляется сейчас естественным даже школьнику, был гораздо более поздним изобретением и потребовал немалой работы. Отдельные списки, датируемые примерно 300 г. до н. э., которые были найдены на островах Эгейского моря, содержат имена членов некоторых религиозных общин, упорядоченные только по первой букве (представляя, таким образом, результат одного прохода поразрядной сортировки слева направо). Некоторые греческие папирусы, датируемые 134–135 г. н. э., содержат списки налогоплательщиков, упорядоченных по первым двум буквам. Аполлоний Софист** (Apollonius Sophista) использовал алфавитный порядок по первым двум буквам (а иногда и по последующим) в своем словаре поэзии Гомера (1 в. н. э.). Известны примеры более совершенного упорядочения, например *Hippocratic Glosses* Галена*** (ок. 200 г.), но это было крайне редким явлением. По первым буквам были упорядочены слова и в *Etymologiarum* Св. Исидора (St. Isidorus)**** (ок. 630 г., книга X), а в *Corpus Glossary* (ок. 725 г.) использовались только две первые буквы каждого слова. Эти две работы были, пожалуй, крупнейшими нечисловыми файлами данных, скомпилированными в средние века.

Впервые описание алфавитного порядка появилось в *Catholicon* Джованни Генуэзского (Giovanni di Genoa's) в 1286 году. В предисловии поясняется, что

amo	предшествует	*bibo*
abeo	предшествует	*adeo*
amatus	предшествует	*amor*
imprudens	предшествует	*impudens*
iusticia	предшествует	*iustus*
polisintheton	предшествует	*polissenus*

(т. е. приводятся примеры ситуаций, в которых порядок следования слов определяется первой, второй, ..., шестой буквами) "и далее точно так же". Он отмечает, что для изобретения подобного способа упорядочения слов потребовались значительные усилия. "Я прошу тебя, читатель, не презирать мой огромный труд и этот порядок, как нечто ничего не стоящее."

Подробным изучением развития алфавитного порядка до начала книгопечатания занимался Ллойд В. Дейли (Lloyd W. Daly) [*Collection Latomus* **90** (1967), 100 p.].

---

* Такая же система была принята в Древней Руси: применялись алфавит и специальный символ ("титло"), а также некоторые дополнительные обозначения для тысяч, миллионов и т. д. — *Прим. перев.*

** Один из грамматиков древности; выходец из Александрии.

*** Римский врач и естествоиспытатель, классик античной медицины.

**** Исидор Севильский — испанский епископ, выдающийся ученый и писатель.

Им найдено несколько интереснейших старинных рукописей, которые, несомненно, использовались в качестве черновых записей при сортировке слов по первой букве (см. с. 89–90 его монографии).

В первом словаре английского языка Роберта Коудри (Robert Cawdrey) *Table Alphabeticall* (London, 1604) содержатся следующие инструкции.

> Если слово, которое ты ищешь, начинается с "a", то ищи его в начале, а если оно начинается с "v", то ищи его в конце книги. Опять-таки, если слово начинается с "ca", то искать его следует в начале буквы "c", а если с "cu", то смотреть надлежит в конце этой буквы. И так поступай до конца слова.

Создается впечатление, что Коудри сам учился своему методу расставления слов в алфавитном порядке в процессе работы — на первых страницах словаря многие слова находятся не на своих местах, в то время как остальная часть словаря содержит гораздо меньше ошибок.

Бинарный поиск был впервые упомянут Джоном Мочли (John Mauchly) в, пожалуй, первой опубликованной дискуссии о нечисленных методах программирования [*Theory and Techniques for the Design of Electronic Digital Computers*, edited by G. W. Patterson, **1** (1946), 9.7–9.8; **3** (1946), 22.8–22.9]. Метод становится хорошо известным программистам, однако ни у кого не возник вопрос, как работать с N, не равным $2^n - 1$. [См. A. D. Booth, *Nature* **176** (1955), 565; A. I. Dumey, *Computers and Automation* **5** (December, 1956), 7 (здесь бинарный поиск назван так: "Двадцать вопросов"); Daniel D. McCracken, *Digital Computer Programming* (Wiley, 1957), 201–203; M. Halpern, *CACM* **1**, 1 (February, 1958), 1–3.]

Д. Г. Лехмер (D. H. Lehmer), пожалуй, был первым, кто опубликовал алгоритм бинарного поиска, работающий при любых N [*Proc. Symp. Appl. Math.* **10** (1960), 180–181]. Следующий шаг был сделан Г. Боттенбруком (H. Bottenbruch), который представил интересный вариант алгоритма B, в котором отдельные проверки равенства отодвигаются в конец алгоритма [*JACM* **9** (1962), 214]. Используя $i \leftarrow \lceil (l+u)/2 \rceil$ вместо $i \leftarrow \lfloor (l+u)/2 \rfloor$ на шаге B2, он устанавливает $l \leftarrow i$ при $K \geq K_i$; тогда $u - l$ уменьшается на каждом шаге. В конце, когда $l = u$, мы имеем $K_l \leq K < K_{l+1}$ и можем проверить, является ли поиск успешным, при помощи еще одного сравнения (Боттенбрук полагал, что изначально $K \geq K_1$). Данная идея позволяет несколько ускорить внутренний цикл на многих компьютерах; этот же принцип может быть применен и к другим алгоритмам поиска, обсуждавшимся в этом разделе. Впрочем, успешный поиск потребует в среднем около одной дополнительной итерации в соответствии с (2). Так как внутренний цикл выполняется примерно около $\lg N$ раз, более быстрый цикл сможет компенсировать дополнительную итерацию только при очень больших N (см. упр. 23).

С другой стороны, алгоритм Боттенбрука будет находить крайнее справа вхождение ключа в таблицу, имеющую дубликаты; а это свойство алгоритма может оказаться важным для определенного класса задач.

К. Ю. Айверсон (K. E. Iverson) [*A Programming Language* (Wiley, 1962), 141] привел описание алгоритма B, однако не рассмотрел случай неудачного поиска. Д. Э. Кнут (D. E. Knuth) [*CACM* **6** (1963), 556–558] представил алгоритм B в качестве примера, использованного автоматической системой построения блок-схем. Однородный бинарный поиск (алгоритм C) был предложен автору А. К. Чандрой (A. K. Chandra) из Станфордского университета в 1971 году.

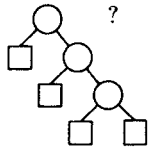
Поиск Фибоначчи изобретен Дэвидом Э. Фергюсоном (David E. Ferguson) [*CACM* **3** (1960), 648]. Бинарные деревья, подобные деревьям Фибоначчи, появились в "пионерской" работе норвежского математика Акселя Тью (Axel Thue) еще в 1910 году (см. упр. 28). Деревья Фибоначчи без меток появились также в качестве курьеза в популярной книге Хьюго Штейнгауза (Hugo Steinhaus) *Mathematical Snapshots* (New York: Stechert, 1938, с. 28). Они изображались корнем вниз и выглядели почти как настоящие деревья, с правыми ветвями, которые были в два раза длиннее левых (так что все листья находились на одном уровне).

Интерполяционный поиск был предложен В. В. Петерсоном (W. W. Peterson) [*IBM J. Res. & Devel.* **1** (1957), 131–132], однако корректный анализ этого метода был сделан значительно позже (см. упр. 22).

УПРАЖНЕНИЯ

▶ **1.** [*21*] Докажите, что если на шаге B2 бинарного поиска $u < l$, то $u = l - 1$ и $K_u < K < K_l$. (Полагаем, что $K_0 = -\infty$ и $K_{N+1} = +\infty$, хотя эти искусственно введенные ключи в действительности алгоритмом не используются и не обязаны находиться в таблице.)

▶ **2.** [*22*] Будет ли алгоритм B продолжать корректно работать при наличии K в таблице, если мы: (a) установим на шаге B5 "$l \leftarrow i$" вместо "$l \leftarrow i + 1$"; (b) установим на шаге B4 "$u \leftarrow i$" вместо "$u \leftarrow i - 1$"; (c) внесем оба эти изменения?

3. [*15*] Какой метод поиска соответствует дереву  ?

Каково среднее число сравнений выполняется при успешном и неудачном поиске?

4. [*20*] Если поиск с использованием программы 6.1S (последовательный поиск) выполняется ровно 638 единиц времени, то как долго будет работать программа B (бинарный поиск)?

5. [*M24*] Для каких значений N программа B в среднем выполняется *медленнее* последовательного поиска (программа 6.1Q′) в случае успешного поиска?

6. [*28*] (К. Ю. Айверсон (K. E. Iverson).) Упр. 5 приводит нас к мысли, что стóит иметь некий гибридный метод, переходящий от бинарного поиска к последовательному при достаточном уменьшении интервала поиска. Напишите соответствующую программу для MIX-компьютера и определите наилучший момент перемены метода поиска.

▶ **7.** [*M22*] Будет ли алгоритм U корректно работать, если мы изменим шаг U1 так, что

a) и i, и m установятся равными $\lfloor N/2 \rfloor$;

b) и i, и m установятся равными $\lceil N/2 \rceil$?

[*Указание.* Предположите, что первый шаг был "Установить $i \leftarrow 0$, $m \leftarrow N$ (или $N + 1$), перейти к шагу U4".]

8. [*M20*] Пусть $\delta_j =$ DELTA[j] является j-м приращением в алгоритме C, как определено в (6).

a) Чему равна сумма $\sum_{j=0}^{\lfloor \lg N \rfloor + 2} \delta_j$?

b) Чему равны минимальное и максимальное значения i, которые могут появиться на шаге C2?

9. [*20*] Существует ли значение $N > 1$, при котором алгоритмы B и C в точности эквивалентны в том смысле, что они оба выполняют одну и ту же последовательность сравнений для всех аргументов поиска?

10. [*21*] Поясните, как написать MIX-программу для алгоритма C, которая будет содержать около $7 \lg N$ инструкций и выполняться примерно за $4.5 \lg N$ единиц времени.

11. [*M26*] Найдите формулы зависимости средних значений $C1$, $C2$ и A от N и S в частотном анализе программы C.

12. [*20*] Изобразите дерево бинарного поиска, соответствующее методу Шара при $N = 12$.

13. [*M24*] Протабулируйте средние значения количества сравнений в методе Шара для $1 \le N \le 16$, рассматривая случаи как успешного, так и неудачного поиска.

14. [*21*] Поясните, как распространить алгоритм F на все значения $N \ge 1$.

15. [*M19*] Для каких значений k дерево Фибоначчи порядка k определяет оптимальную процедуру поиска (имеется в виду наименьшее среднее количество выполняемых сравнений)?

16. [*21*] На рис. 9 показана диаграмма размножения кроликов в исходной задаче Фибоначчи (см. раздел 1.2.8). Существует ли простая взаимосвязь между этой диаграммой и обсуждавшимся в разделе деревом Фибоначчи?

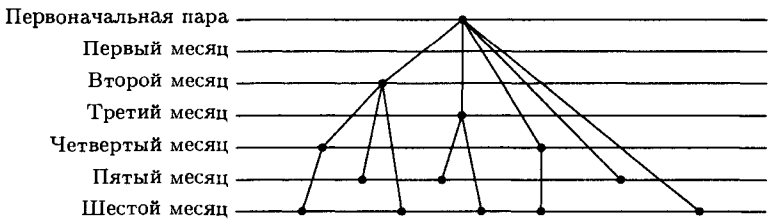


Рис. 9. Пары кроликов, размножающиеся в соответствии с правилом Фибоначчи.

17. [*M21*] Из упр. 1.2.8–34 или 5.4.2–10 известно, что каждое натуральное число n единственным образом представляется в виде суммы чисел Фибоначчи $n = F_{a_1} + F_{a_2} + \cdots + F_{a_r}$, где $r \ge 1$, $a_j \ge a_{j+1} + 2$ при $1 \le j < r$ и $a_r \ge 2$. Докажите, что в дереве Фибоначчи порядка k путь от корня до узла (n) имеет длину $k + 1 - r - a_r$.

18. [*M30*] Найдите точные формулы зависимостей средних значений $C1$, $C2$ и A от k, F_k, F_{k+1} и S при частотном анализе программы F.

19. [*M42*] Проведите детальный анализ среднего времени работы алгоритма, предложенного в упр. 14.

20. [*M22*] Количество сравнений, выполняемых при бинарном поиске, приблизительно равно $\log_2 N$, а при поиске Фибоначчи — $(\phi/\sqrt{5}) \log_\phi N$. Цель этого упражнения — показать, что данные формулы представляют собой частные случаи более общего результата.

Пусть p и q — положительные числа и $p + q = 1$. Рассмотрим алгоритм поиска в таблице из N чисел в порядке возрастания. Алгоритм начинается со сравнения аргумента с (pN)-м ключом и затем повторяет эту процедуру с меньшими блоками, выбираемыми исходя из результатов сравнения. (Бинарный поиск соответствует $p = q = 1/2$; поиск Фибоначчи — $p = 1/\phi$, $q = 1/\phi^2$.)

Обозначим среднее количество сравнений, необходимых для поиска в таблице размером N через $C(N)$. Оно приближенно удовлетворяет следующим соотношениям:

$$C(1) = 0; \qquad C(N) = 1 + pC(pN) + qC(qN) \quad \text{для } N > 1.$$

Это можно объяснить тем, что после сравнения поиск сводится примерно с вероятностью p к поиску среди pN элементов и с вероятностью q — к поиску среди qN элементов. При

больших N мы можем пренебречь эффектами, связанными с тем, что pN и qN — не целые числа.

a) Покажите, что $C(N) = \log_b N$ удовлетворяет приведенным соотношениям при некотором b (для бинарного поиска и поиска Фибоначчи величина b согласуется с приведенными формулами).

b) Рассмотрим следующее рассуждение: "С вероятностью p интервал поиска делится на $1/p$ и с вероятностью q — на $1/q$. Таким образом, интервал в среднем делится на $p\cdot(1/p)+q\cdot(1/q) = 2$, т. е. алгоритм при любых p и q одинаково хорош, в точности как и алгоритм бинарного поиска". Содержится ли в приведенном рассуждении ошибка?

21. [*20*] Изобразите бинарное дерево для интерполяционного поиска при $N = 10$.

22. [*M41*] (Э. Ч. Яо (A. C. Yao) и Ф. Ф. Яо (F. F. Yao).) Покажите, что корректно реализованный интерполяционный поиск асимптотически требует в среднем $\lg\lg N$ сравнений в случае N рассортированных независимых равномерно распределенных ключей. Более того, *все* алгоритмы поиска в таких таблицах должны в среднем выполнять $\lg\lg N$ сравнений (асимптотическая оценка).

▶ **23.** [*25*] Алгоритм бинарного поиска Боттенбрука, упоминавшийся в конце этого раздела, позволяет избегать проверок равенства до конца поиска (при работе алгоритма нам известно, что $K_l \le K < K_{u+1}$ и проверка равенства не проводится до тех пор, пока не будет выполнено условие $l = u$). Такой трюк мог бы немного ускорить работу программы B для больших N, так как команда JE была бы удалена из внутреннего цикла. (Данная идея почти не реальна в связи с тем, что $\lg N$ обычно достаточно мал, а дополнительная итерация в нашей программе компенсируется только при $N > 2^{66}$ для успешного поиска, так как время работы "уменьшается" с $(18\lg N - 16)u$ до $(17.5\lg N + 17)u$!)

Покажите, что *любой* алгоритм поиска, соответствующий бинарному дереву и использующий тройное ветвление во внутренних узлах ($<$, $=$ или $>$), может быть преобразован в алгоритм с двойным ветвлением ($<$ или $\ge$). В частности, покажите, каким образом можно модифицировать алгоритм C.

▶ **24.** [*23*] В разделах 2.3.4.5 и 5.2.3 мы видели, что полное бинарное дерево является удобным способом представления дерева с минимальной длиной пути в последовательных ячейках. Разработайте эффективный метод поиска, основанный на таком представлении. [*Указание.* Можно ли в бинарном поиске использовать вместо деления на 2 умножение на 2?]

▶ **25.** [*M25*] Предположим, что бинарное дерево имеет a_k внутренних и b_k внешних узлов на уровне k ($k = 0, 1, \ldots$; корень находится на нулевом уровне). Так, на рис. 8 имеем $(a_0, a_1, \ldots, a_5) = (1, 2, 4, 4, 1, 0)$ и $(b_0, b_1, \ldots, b_5) = (0, 0, 0, 4, 7, 2)$.

a) Покажите, что существует простое алгебраическое соотношение между производящими функциями $A(z) = \sum_k a_k z^k$ и $B(z) = \sum_k b_k z^k$.

b) Распределение вероятностей для успешного поиска в бинарном дереве имеет производящую функцию $g(z) = zA(z)/N$, а для неудачного — $h(z) = B(z)/(N+1)$. (Учитывая используемые в разделе обозначения, можно записать: $C_N = \mathrm{mean}(g)$, $C'_N = \mathrm{mean}(h)$; выражение (2) связывает эти величины.) Найдите зависимость между $\mathrm{var}(g)$ и $\mathrm{var}(h)$.

26. [*22*] Покажите, что деревья Фибоначчи связаны с сортировкой методом многофазного слияния на трех лентах.

27. [*M30*] (Г. С. Стоун (H. S. Stone) и Джон Линн (John Linn).) Рассмотрим процесс поиска, в котором одновременно используется k процессоров и который базируется на сравнении ключей. На каждом шаге поиска определяется k индексов — $i_1, \ldots, i_k$ — и выполняется k одновременных сравнений. Если для некоторого j выполняется $K = K_{i_j}$, поиск завершается успешно. В противном случае мы переходим к следующему шагу, основанному на 2^k возможных исходах сравнений: $K < K_{i_j}$ или $K > K_{i_j}$, $1 \le j \le k$.

Докажите, что такой процесс при $N \to \infty$ должен выполнять в среднем минимум $\log_{k+1} N$ шагов (в предположении, что все ключи таблицы, как и аргумент поиска, равновероятны). (Следовательно, выигрыш от использования k процессоров составляет не k раз, как может показаться, а только $\lg(k+1)$ раз; в этом смысле выгоднее выполнять поиск с помощью одного отдельного процессора, не кооперируя их.)

28. [*M23*] Определим *дерево Тью (Thue)* T_n при помощи алгебраического выражения с использованием бинарного оператора "$*$" следующим образом: $T_0(x) = x * x$, $T_1(x) = x$, $T_{n+2}(x) = T_{n+1}(x) * T_n(x)$.

a) Количество листьев дерева T_n равно количеству x в полной записи $T_n(x)$. Выразите его при помощи чисел Фибоначчи.

b) Докажите, что если бинарный оператор "$*$" удовлетворяет аксиоме

$$((x * x) * x) * ((x * x) * x) = x\,,$$

то $T_m(T_n(x)) = T_{m+n-1}(x)$ для всех $m \geq 0$ и $n \geq 1$.

▶ **29.** [*22*] (Поль Фельдман (Paul Feldman), 1985.) Вместо предположения, что $K_1 < K_2 < \cdots < K_N$, допустим, что $K_{p(1)} < K_{p(2)} < \cdots < K_{p(N)}$, где перестановка $p(1)p(2)\ldots p(N)$ является инволюцией* и $p(j) = j$ для всех четных значений j. Покажите, что можно найти любой данный ключ K или выяснить, что он отсутствует, проведя максимум $2\lfloor \lg N \rfloor + 1$ сравнений.

30. [*27*] (*Инволюционное кодирование.*) Используя идеи предыдущего упражнения, расположите N ключей таким образом, чтобы их относительный порядок неявно кодировал произвольно взятый массив t-битовых чисел $x_1, x_2, \ldots, x_m$; $m \leq N/4 + 1 - 2^t$. Найденный вами способ расположения должен позволять определять первые k бит числа x_j при помощи k сравнений для любого j, а также находить произвольный ключ при помощи $\leq 2\lfloor \lg N \rfloor + 1$ сравнений. (Этот результат используется в теоретическом изучении структур данных, асимптотически эффективных во времени и пространстве.)

6.2.2. Поиск по бинарному дереву

В предыдущем разделе мы видели, что неявная структура бинарного дерева облегчает понимание бинарного поиска и поиска Фибоначчи. Рассмотрев соответствующие деревья, мы показали, что для заданного значения N количество сравнений, необходимое для бинарного поиска, не превышает количества, которое требуется для любого другого метода, основанного на сравнении ключей. Однако методы из предыдущего раздела годятся, в основном, для таблиц фиксированного размера, поскольку последовательное расположение записей делает вставку и удаление записей весьма трудоемкими. Если таблица динамически изменяется, то мы можем потратить на ее постоянное упорядочение куда больше времени, чем сэкономить на бинарном поиске.

Явное использование структуры бинарного дерева позволяет быстро вставлять и удалять записи, а также эффективно выполнять поиск в таблице. В результате мы получаем метод, эффективный как для поиска, так и для сортировки. Цена такой гибкости — два поля ссылок в каждой записи таблицы.

Технологии поиска в растущей таблице часто называют *алгоритмами таблиц символов*, так как ассемблеры, компиляторы и другие системные программы обычно применяют их для хранения пользовательских символов. Например, ключом записи

---

* Инволюцией называется такое отображение некоторого множества на себя, квадрат которого является тождественным отображением. — *Прим. перев.*

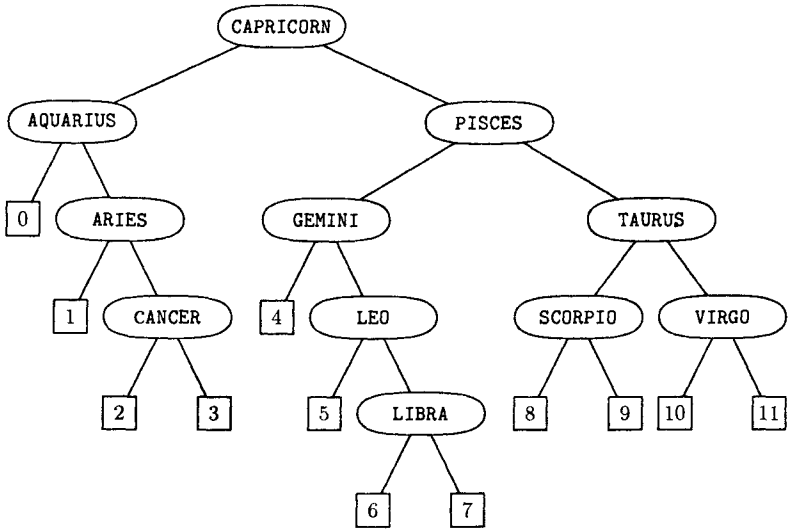


Рис. 10. Бинарное дерево поиска.

в компиляторе может быть символьный идентификатор переменной в некоторой программе на языке FORTRAN или C, а остальная часть записи может содержать информацию о типе переменной и ее адресе; другим примером служит ключ, являющийся символом в MIX-программе, а остаток записи содержит его эквивалент. Программы поиска со вставкой по дереву, которые будут описаны в этом разделе, достаточно эффективны в качестве алгоритмов таблиц символов, в особенности в приложениях, в которых может оказаться желательным вывод списка символов в алфавитном порядке. Другие алгоритмы таблиц символов описываются в разделах 6.3 и 6.4.

На рис. 10 показано бинарное дерево поиска, содержащее названия одиннадцати знаков зодиака*. Если мы приступим к поиску двенадцатого имени, SAGITTARIUS, начиная от корня дерева, то увидим, что оно больше, чем CAPRICORN, и нужно идти вправо; оно больше, чем PISCES, а потому мы опять идем вправо. Это имя меньше, чем TAURUS, поэтому теперь мы сворачиваем влево. Так как искомое имя меньше, чем SCORPIO, мы попадаем во внешний узел $\boxed{8}$. Поиск оказался неудачным, и теперь мы можем *вставить* SAGITTARIUS в то место, где завершился наш поиск, на место внешнего узла $\boxed{8}$. Таким образом, таблица может расширяться без перемещения существующих записей. Дерево на рис. 10 получено путем последовательной вставки в пустое дерево ключей CAPRICORN, AQUARIUS, PISCES, ARIES, TAURUS, GEMINI, CANCER, LEO, VIRGO, LIBRA, SCORPIO в указанном порядке.

На рис. 10 все ключи левого поддерева меньше (в алфавитном порядке), чем CAPRICORN, а все ключи правого поддерева — больше. То же самое справедливо и для левого и правого поддеревьев любого узла. Отсюда следует, что при обходе дерева в *симметричном порядке* (см. раздел 2.3.1), поскольку симметричный порядок

* Список знаков зодиака, упорядоченный по месяцам, таков: Козерог (Capricorn), Водолей (Aquarius), Рыбы (Pisces), Овен (Aries), Телец (Taurus), Близнецы (Gemini), Рак (Cancer), Лев (Leo), Дева (Virgo), Весы (Libra), Скорпион (Scorpio), Стрелец (Sagittarius). — *Прим. перев.*

основан на обходе сначала левого поддерева каждого узла, затем — самого узла, а после — правого поддерева, мы получим список ключей в алфавитном порядке:

AQUARIUS, ARIES, CANCER, CAPRICORN, GEMINI, LEO, ..., VIRGO.

Ниже приводится подробное описание процесса поиска со вставкой.

Алгоритм Т (*Поиск по дереву со вставкой (Tree search and insertion)*). Дана таблица записей, образующая бинарное дерево. Этот алгоритм (рис. 11) предназначен для поиска данного аргумента K. Если K в таблице отсутствует, новый узел, содержащий K, вставляется в дерево в надлежащее место.

Предполагается, что узлы дерева содержат, по крайней мере, следующие поля:

KEY(P) — ключ, хранящийся в узле NODE(P);

LLINK(P) — указатель на левое поддерево узла NODE(P);

RLINK(P) — указатель на правое поддерево узла NODE(P).

Пустые поддеревья (внешние узлы на рис. 10) представляются пустыми указателями Λ. Переменная ROOT указывает на корень дерева. Для удобства полагаем, что дерево не пусто, т. е. ROOT $\neq \Lambda$, так как при ROOT $= \Lambda$ необходимые операции тривиальны.

T1. [Инициализация.] Установить P $\leftarrow$ ROOT. (Переменная-указатель P будет перемещаться вниз по дереву.)

T2. [Сравнение.] Если $K <$ KEY(P), перейти к шагу T3; если $K >$ KEY(P), перейти к шагу T4; и если $K =$ KEY(P), поиск успешно завершается.

T3. [Перемещение влево.] Если LLINK(P) $\neq \Lambda$, установить P $\leftarrow$ LLINK(P) и перейти к шагу T2; в противном случае перейти к шагу T5.

T4. [Перемещение вправо.] Если RLINK(P) $\neq \Lambda$, установить P $\leftarrow$ RLINK(P) и перейти к шагу T2.

T5. [Вставка в дерево.] (Поиск оказался неудачным; теперь мы должны поместить K в дерево.) Установить Q $\Leftarrow$ AVAIL (адрес нового узла). Присвоить KEY(Q) $\leftarrow$ K, LLINK(Q) $\leftarrow$ RLINK(Q) $\leftarrow \Lambda$. (На практике следует инициализировать и другие поля нового узла.) Если K был меньше, чем KEY(P), следует назначить LLINK(P) $\leftarrow$ Q; в противном случае — назначить RLINK(P) $\leftarrow$ Q. (Теперь мы можем присвоить P $\leftarrow$ Q и считать поиск успешно завершившимся.) ▮

Этот алгоритм сам подсказывает нам свою программную реализацию. Предположим, например, что узлы дерева имеют структуру

+	0	LLINK	RLINK
KEY			

, (1)

за которой, возможно, следует дополнительная информация INFO. Используя, как и в главе 2, список свободной памяти AVAIL, мы можем написать следующую MIX-программу.

Программа Т (*Поиск по дереву со вставкой*). rA $\equiv K$, rI1 $\equiv$ P, rI2 $\equiv$ Q.

```
01  LLINK EQU  2:3
02  RLINK EQU  4:5
```

```
03  START LDA  K              1          T1. Инициализация.
04        LD1  ROOT           1          P ← ROOT.
05        JMP  2F             1
06  4H    LD2  0,1(RLINK)     C2         T4. Перемещение вправо. Q ← RLINK(P).
07        J2Z  5F             C2         Переход к шагу T5 при Q = Λ.
08  1H    ENT1 0,2            C − 1      P ← Q.
09  2H    CMPA 1,1            C          T2. Сравнение.
10        JG   4B             C          Переход к шагу T4 при K > KEY(P).
11        JE   SUCCESS        C1         Выход, если K = KEY(P).
12        LD2  0,1(LLINK)     C1 − S     T3. Перемещение влево. Q ← LLINK(P).
13        J2NZ 1B             C1 − S     Переход к шагу T2 при Q ≠ Λ.
14  5H    LD2  AVAIL          1 − S      T5. Вставка в дерево.
15        J2Z  OVERFLOW       1 − S
16        LDX  0,2(RLINK)     1 − S
17        STX  AVAIL          1 − S      Q ⇐ AVAIL.
18        STA  1,2            1 − S      KEY(Q) ← K.
19        STZ  0,2            1 − S      LLINK(Q) ← RLINK(Q) ← Λ.
20        JL   1F             1 − S      Выполнялось ли условие K < KEY(P)?
21        ST2  0,1(RLINK)     A          RLINK(P) ← Q.
22        JMP  *+2            A
23  1H    ST2  0,1(LLINK)     1 − S − A  LLINK(P) ← Q.
24  DONE  EQU  *              1 − S      Выход после вставки. ▌
```

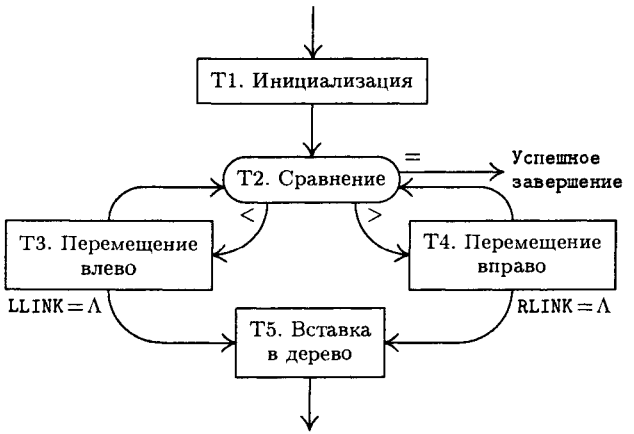


Рис. 11. Поиск по дереву и вставка.

Первые 13 строк программы выполняют поиск, следующие 11 — вставку. Время работы фазы поиска составляет $(7C + C1 - 3S + 4)u$, где

C — количество выполненных сравнений;

$C1$ — количество сравнений, когда $K \le$ KEY(P);

$C2$ — количество сравнений, когда $K >$ KEY(P);

S — 1 при успешном и 0 при неудачном поиске.

В среднем имеем $C1 = \frac{1}{2}(C + S)$, поскольку $C1 + C2 = C$, а $C1 - S$ имеет то же распределение вероятностей, что и $C2$; таким образом, время работы программы —

около $(7.5C - 2.5S + 4)u$. Как видим, это лучшее значение, чем при бинарном поиске с неявными деревьями (см. программу 6.2.1C). Продублировав код, как это было сделано в программе 6.2.1F, мы можем избавиться от строки 08 программы T, уменьшая время работы до $(6.5C - 2.5S + 5)u$. При неудачном поиске фаза вставки займет дополнительное время — $14u$ или $15u$.

Алгоритм T легко адаптировать для работы с ключами и записями *переменной длины*. Например, если мы распределяем имеющуюся память последовательно, по принципу LIFO ("последним вошел — первым вышел"), можно очень легко создавать узлы различных размеров — первое слово в (1) может указывать размер записи. Такое достаточно эффективное использование памяти делает алгоритмы таблиц символов, основанные на деревьях, особенно привлекательными для использования в компиляторах, ассемблерах и загрузчиках.

А как насчет наихудшего варианта? Программисты зачастую скептически относятся к алгоритму T при первом знакомстве с ним. Ведь если бы ключи из рис. 10 поступали в алфавитном порядке — AQUARIUS, ..., VIRGO — вместо календарного, алгоритм построил бы вырожденное дерево, соответствующее *последовательному* поиску. Все поля LLINK в этом дереве были бы пустыми. Точно так же поступление ключей в несколько необычном порядке —

AQUARIUS, VIRGO, ARIES, TAURUS, CANCER, SCORPIO,
CAPRICORN, PISCES, GEMINI, LIBRA, LEO —

привело бы к построению зигзагообразного дерева, что ничуть не сокращает время поиска (постройте его сами — и убедитесь).

С другой стороны, для успешного поиска по дереву, изображенному на рис. 10, требуется в среднем всего $3\frac{2}{11}$ сравнений; это лишь ненамного превышает минимально возможное количество 3, которое достигается при поиске по наилучшему из возможных бинарных деревьев.

При работе с хорошо сбалансированным деревом время поиска примерно пропорционально $\log N$; однако в случае вырожденного дерева время поиска становится пропорциональным N. В упр. 2.3.4.5–5 доказывается, что среднее время поиска пропорционально $\sqrt{N}$ в предположении равновероятности всех бинарных деревьев с N узлами. Чего же нам следует ожидать от алгоритма T?

К счастью, можно доказать, что поиск по дереву требует лишь около $2\ln N \approx 1.386\lg N$ сравнений, если ключи вставляются в дерево в случайном порядке. На практике в основном встречаются хорошо сбалансированные деревья, в то время как вырожденные деревья появляются редко.

Доказать это утверждение на удивление легко. Предположим, что каждая из $N!$ возможных перестановок N ключей с равной вероятностью используется для построения дерева путем последовательных вставок. Число сравнений, необходимых для поиска ключа, ровно на одно больше числа сравнений, необходимых для его вставки в дерево. Таким образом, если C_N — среднее количество сравнений при успешном поиске, а C'_N — при неудачном, то

$$C_N = 1 + \frac{C'_0 + C'_1 + \cdots + C'_{N-1}}{N}. \qquad (2)$$

Однако согласно соотношению между длинами внутреннего и внешнего путей (6.2.1–(2))

$$C_N = \left(1 + \frac{1}{N}\right) C'_N - 1. \tag{3}$$

Объединяя (3) и (2), получим

$$(N+1)C'_N = 2N + C'_0 + C'_1 + \cdots + C'_{N-1}. \tag{4}$$

Это рекуррентное соотношение легко решается: вычтя следующее равенство

$$NC'_{N-1} = 2(N-1) + C'_0 + C'_1 + \cdots + C'_{N-2}$$

из предыдущего, мы получим

$$(N+1)C'_N - NC'_{N-1} = 2 + C'_{N-1}, \qquad \text{следовательно,} \qquad C'_N = C'_{N-1} + 2/(N+1).$$

Так как $C'_0 = 0$, то

$$C'_N = 2H_{N+1} - 2. \tag{5}$$

Применив (3) и выполнив некоторые упрощения, получим требуемый результат:

$$C_N = 2\left(1 + \frac{1}{N}\right) H_N - 3. \tag{6}$$

В упр. 6–8 приводится более детальная информация — можно определить не только средние значения C_N и C'_N, но и их точные распределения вероятностей.

Сортировка путем вставки в дерево. Алгоритм T был разработан для поиска, однако он может применяться и в качестве основы алгоритма внутренней *сортировки*; его можно рассматривать как естественное обобщение вставки в список — алгоритма 5.2.1L. Если этот алгоритм будет корректно реализован, то среднее время его работы лишь незначительно превысит время работы лучших алгоритмов, обсуждавшихся в главе 5. После того как дерево построено, остается только симметрично его обойти (алгоритм 2.3.1T); так мы получим записи в рассортированном виде.

Тем не менее следует предпринять определенные меры предосторожности — требуются некоторые отличия на шаге T2 при $K = \texttt{KEY(P)}$ в связи с тем, что нашей целью является не поиск, а сортировка. Одно из возможных решений — считать, что $K = \texttt{KEY(P)}$ эквивалентно $K > \texttt{KEY(P)}$. Сортировка при этом будет вполне корректна (отметим, что записи с одинаковыми ключами необязательно должны находиться в смежных узлах — важно, чтобы они последовательно встречались при симметричном обходе дерева). Однако при наличии большого количества повторяющихся ключей этот метод может дать плохо сбалансированное дерево, и процесс сортировки окажется замедленным. Еще одним решением может служить хранение списка записей с одним и тем же ключом для каждого из узлов. В этом случае потребуется дополнительное поле ссылок, однако при большом числе повторяющихся ключей такая сортировка окажется более быстрой.

Таким образом, применение алгоритма T только для сортировки является неплохим, хотя и не лучшим, решением. Однако этот алгоритм настоятельно рекомендуется использовать в случае комбинирования в одном приложении как поиска, так и сортировки.

Интересно отметить тесную взаимосвязь между анализом сортировки путем вставки в дерево и анализом "быстрой сортировки", хотя, на первый взгляд, эти

методы совершенно различны. При успешной вставке N ключей в изначально пустое дерево мы выполняем то же среднее число сравнений ключей, что и в алгоритме 5.2.2Q (за небольшими исключениями). Например, в случае вставки в дерево каждый ключ сравнивается с K_1, затем каждый ключ, меньший, чем K_1, сравнивается с первым ключом, меньшим, чем K_1, и т. д.; в случае быстрой сортировки каждый ключ сравнивается с первым разделяющим элементом K, затем каждый ключ, меньший, чем K, сравнивается с некоторым элементом, меньшим, чем K, и т. д. Среднее количество сравнений, необходимое в обоих случаях, равно $NC_N - N$ (впрочем, в алгоритме 5.2.2Q на самом деле выполняется несколько дополнительных сравнений для ускорения внутреннего цикла).

Удаление. Иногда приходится заставлять компьютер забыть одну из записей таблицы. Мы можем легко удалить узел, поле LLINK или RLINK которого равно Λ; однако, если оба поддерева не пусты, мы должны проделать некоторые специальные действия, так как ссылка не может указывать на два места одновременно.

Рассмотрим еще раз рис. 10. Как удалить из дерева корневой узел, CAPRICORN? Одно из решений состоит в удалении *следующего* по алфавиту узла, ссылка LLINK которого пуста, и его вставке на место узла, который надо удалить. Например, на рис. 10 мы можем удалить GEMINI, а затем заменить CAPRICORN значением GEMINI. Это позволит сохранить порядок элементов таблицы. В приведенном далее алгоритме содержится детальное описание такого процесса.

Алгоритм D (*Удаление из дерева (Tree deletion)*). Пусть Q — переменная, указывающая на узел дерева поиска, представленного, как в алгоритме T. Алгоритм предназначен для удаления этого узла, причем дерево останется бинарным деревом поиска. (На практике имеем либо Q $\equiv$ ROOT, либо Q $\equiv$ LLINK(P), либо Q $\equiv$ RLINK(P) для некоторого узла дерева. Алгоритм изменяет значение Q с учетом проведенного удаления.)

D1. [Пуста ли ссылка RLINK?] Установить T $\leftarrow$ Q. Если RLINK(T) $= \Lambda$, установить Q $\leftarrow$ LLINK(T) и перейти к шагу D4. (Например, если Q $\equiv$ RLINK(P) для некоторого P, мы можем присвоить RLINK(P) $\leftarrow$ LLINK(T).)

D2. [Поиск преемника.] Установить R $\leftarrow$ RLINK(T). Если LLINK(R) $= \Lambda$, присвоить LLINK(R) $\leftarrow$ LLINK(T), Q $\leftarrow$ R и перейти к шагу D4.

D3. [Поиск пустой ссылки LLINK.] Установить S $\leftarrow$ LLINK(R). Затем, если LLINK(S) $\neq \Lambda$, присвоить R $\leftarrow$ S и повторять этот шаг до тех пор, пока не будет получено LLINK(S) $= \Lambda$. (Теперь S будет равно Q\$, следующему за Q элементу при симметричном обходе.) И, наконец, установить LLINK(S) $\leftarrow$ LLINK(T), LLINK(R) $\leftarrow$ RLINK(S), RLINK(S) $\leftarrow$ RLINK(T), Q $\leftarrow$ S.

D4. [Освобождение узла.] Выполнить AVAIL $\Leftarrow$ T, т. е. вернуть удаленный узел в пул свободной памяти. ▌

Читатель может проверить работоспособность данного алгоритма, удаляя узлы AQUARIUS, CANCER и CAPRICORN на рис. 10; каждый из этих случаев имеет свои особенности. Особо придирчивый читатель, конечно, обратит внимание на то, что случай RLINK(T) $\neq \Lambda$, LLINK(T) $= \Lambda$ отдельно не выделен. Мы отложили обсуждение этого специального случая "на потом", поскольку в приведенном виде алгоритм обладает некоторыми интересными свойствами.

Алгоритм D представляется весьма несимметричным, и создается впечатление, что длинная цепь удалений разбалансирует дерево (и все оценки эффективности работы при этом потеряют силу). Однако "все не так, как на самом деле", и вырождения не будет.

Теорема H (Т. Н. Хиббард (T. N. Hibbard), 1962.) *После удаления элемента из случайного дерева по алгоритму D дерево остается случайным.*

[Читателям, которым математика не доставляет никакого удовольствия, настоятельно рекомендуется пропустить все до формулы (10).] Эта формулировка, естественно, несколько туманна. Сформулируем теорему более строго.
Пусть T представляет собой дерево из n элементов и пусть $P(T)$ — вероятность появления T, если его ключи вставляются в случайном порядке согласно алгоритму T. Одни деревья более вероятны, чем другие. Пусть $Q(T)$ — вероятность того, что после вставки случайным образом с помощью алгоритма T $n+1$ элементов и удаления любого из них (выбранного наудачу) по алгоритму D получится дерево T. При вычислении $P(T)$ предполагается, что все $n!$ перестановок ключей равновероятны; при вычислении $Q(T)$ предполагается равновероятность $(n+1)!\,(n+1)$ перестановок ключей и выбора удаляемого ключа. Теорема гласит, что для всех T $P(T) = Q(T)$.

Доказательство. По условию теоремы равновероятны не деревья, а перестановки, поэтому доказывать теорему мы будем, рассматривая в качестве случайных объектов *перестановки*. Сначала определим удаление из перестановки, а затем докажем, что *случайный элемент, удаленный из случайной перестановки, оставляет перестановку случайной.*

Пусть $a_1\,a_2\ldots a_{n+1}$ — перестановка множества $\{1,2,\ldots,n+1\}$; мы хотим определить операцию удаления a_i, которая даст в результате перестановку $b_1\,b_2\ldots b_n$ множества $\{1,2,\ldots,n\}$. Эта операция должна соответствовать алгоритмам T и D, так что, если мы начнем с дерева, сконструированного последовательностью вставок $a_1, a_2, \ldots, a_{n+1}$, и удалим a_i, перенумеровав затем ключи от 1 до n, получится дерево, которое может быть построено последовательностью вставок $b_1\,b_2\ldots b_n$.

Такую операцию удаления можно легко определить. Возможны два случая.

Случай 1. $a_i = n+1$ или $a_i + 1 = a_j$ для некоторого $j < i$. (Это соответствует условию $\mathtt{RLINK}(a_i) = \Lambda$.) Удаляем a_i из последовательности и вычитаем единицу из каждого элемента, большего, чем a_i.

Случай 2. $a_i + 1 = a_j$ для некоторого $j > i$. Заменим a_i элементом a_j, удалим a_j из его первоначального положения и вычтем единицу из каждого элемента, большего, чем a_i.

Рассмотрим, например, перестановку 4 6 1 3 5 2. Если пометить элемент, который должен быть удален, кружком, получим

$$
\begin{array}{llll}
\textcircled{4}\,6\;1\;3\;5\;2 = 4\;5\;1\;3\;2 & & 4\;6\;1\,\textcircled{3}\,5\;2 = 3\;5\;1\;4\;2 \\
4\,\textcircled{6}\,1\;3\;5\;2 = 4\;1\;3\;5\;2 & & 4\;6\;1\;3\,\textcircled{5}\,2 = 4\;5\;1\;3\;2\,. \\
4\;6\,\textcircled{1}\,3\;5\;2 = 3\;5\;1\;2\;4 & & 4\;6\;1\;3\;5\,\textcircled{2} = 3\;5\;1\;2\;4
\end{array}
$$

Поскольку имеется $(n+1)!\,(n+1)$ возможных операций удаления, теорема будет доказана, если мы сможем показать, что каждая перестановка $\{1,2,\ldots,n\}$ является результатом в точности $(n+1)^2$ удалений.

Пусть $b_1\, b_2 \ldots b_n$ — перестановка множества $\{1, 2, \ldots, n\}$. Мы определим $(n+1)^2$ удалений, по одному для каждой пары i, j $(1 \le i, j \le n+1)$.

Удаление для $i < j$ таково:

$$b'_1 \ldots b'_{i-1}\, \textcircled{b_i}\, b'_{i+1} \ldots b'_{j-1}\, (b_i{+}1)\, b'_j \ldots b'_n. \tag{7}$$

Здесь и далее b'_k означает либо b_k, либо $b_k + 1$, в зависимости от того, является ли b_k меньше, чем помеченный элемент. Такое удаление соответствует случаю 2.

Удаление для $i > j$ таково:

$$b'_1 \ldots b'_{i-1}\, \textcircled{b_j}\, b'_i \ldots b'_n; \tag{8}$$

оно соответствует случаю 1.

И наконец, если $i = j$, мы имеем другой случай 1, а именно

$$b'_1 \ldots b'_{i-1}\, \textcircled{n{+}1}\, b'_i \ldots b'_n. \tag{9}$$

В качестве примера примем $n = 4$ и рассмотрим 25 удалений, приводящих к перестановке 3 1 4 2.

	$i=1$	$i=2$	$i=3$	$i=4$	$i=5$
$j=1$	⑤ 3 1 4 2	4 ③ 1 5 2	4 1 ③ 5 2	4 1 5 ③ 2	4 1 5 2 ③
$j=2$	③ 4 1 5 2	3 ⑤ 1 4 2	4 2 ① 5 3	4 2 5 ① 3	4 2 5 3 ①
$j=3$	③ 1 4 5 2	4 ① 2 5 3	3 1 ⑤ 4 2	3 1 5 ④ 2	3 1 5 2 ④
$j=4$	③ 1 5 4 2	4 ① 5 2 3	3 1 ④ 5 2	3 1 4 ⑤ 2	4 1 5 3 ②
$j=5$	③ 1 5 2 4	4 ① 5 3 2	3 1 ④ 2 5	4 1 5 ② 3	3 1 4 2 ⑤

Помеченный элемент всегда находится в i-й позиции; для фиксированного i мы строили $n+1$ различных удалений, по одному для каждого j. Таким образом, для каждой перестановки $b_1\, b_2 \ldots b_n$ построено $(n+1)^2$ различных удалений. Так как возможны только $(n+1)^2 n!$ удалений, мы должны найти их все. ▌

Доказательство теоремы H не только описывает результат удалений, но также помогает проанализировать среднее время удаления. В упр. 12 показано, что при удалении случайного элемента из случайной таблицы шаг D2 будет выполняться меньше чем в половине случаев.

Теперь рассмотрим, как часто выполняется цикл на шаге D3. Предположим, что мы удаляем узел на уровне l и *внешний* узел, при симметричном обходе следующий непосредственно за ним, находится на уровне k. Например, при удалении узла CAPRICORN на рис. 10 $l = 0$ и $k = 3$, так как узел $\boxed{4}$ находится на уровне 3. Если $k = l + 1$, то на шаге D1 RLINK(T) $= \Lambda$; если $k > l + 1$, то на шаге D3 мы выполним присвоение S $\leftarrow$ LLINK(R) ровно $k - l - 2$ раз. Среднее значение l равно (длина внутреннего пути)/N; среднее значение k —

(длина внешнего пути — расстояние до крайнего слева внешнего узла)/N.

Расстояние до крайнего слева внешнего узла равно числу лево-правых минимумов вставляемой последовательности, так что среднее значение этой величины согласно анализу в разделе 1.2.10 составляет H_N. Поскольку длина внешнего пути превышает длину внутреннего пути на $2N$, среднее значение $k - l - 2$ равно $-H_N/N$. Добавляя

к этому среднему значению число случаев, когда $k-l-2$ равно -1, мы получим, что при случайном удалении в *среднем операция* $S \leftarrow$ `LLINK(R)` *на шаге D3 выполняется только*

$$\tfrac{1}{2} + \left(\tfrac{1}{2} - H_N\right)/N \tag{10}$$

раз. Это успокаивает, так как в наихудшем случае, как показано в упр. 11, удаления выполняются весьма медленно.

Хотя теорема H вполне строга и точна, в указанном виде ее *нельзя* применить к последовательности удалений, следующих за вставками. После удалений форма дерева остается случайной, однако относительное распределение значений в данной форме дерева может оказаться измененным, и это может привести к тому, что первая же вставка после удаления *уничтожит* свойство случайности формы. Этот поразительный факт, впервые обнаруженный Гарри Кноттом (Gary Knott) в 1972 году, вам придется принять на веру (см. упр. 15). Еще более удивительны эмпирические данные, накопленные Дж. Л. Эппингером (J. L. Eppinger) (*CACM* **26** (1983), 663–669, **27** (1984), 235), который обнаружил, что длина пути немного уменьшается при небольшом количестве удалений и вставок, однако затем начинает *увеличиваться* до тех пор, пока не достигнет стабильного состояния после выполнения примерно n^2 операций удаления/вставки. Это стабильное состояние *хуже* поведения случайного дерева при N, большем примерно 150. Дальнейшее исследование Кульберсона (Culberson) и Мунро (Munro) (*Comp. J.* **32** (1989), 68–75; *Algorithmica* **5** (1990), 295–311) привело к правдоподобному предположению о том, что среднее время поиска в стабильном состоянии асимптотически равно $\sqrt{2N/9\pi}$. Однако Эппингер предложил простую модификацию, при которой алгоритм D и его зеркальное отражение чередуются в одном алгоритме; он обнаружил, что это приводит к отличному стабильному состоянию, в котором длина пути снижается до примерно 88% от его обычного значения для случайных деревьев. Теоретического пояснения этого факта пока не найдено.

Как упоминалось ранее, алгоритм D не проверяет случай `LLINK(T)` $= \Lambda$, хотя это один из простейших случаев удаления. Мы можем добавить новый шаг D$1\frac{1}{2}$ между шагами D1 и D2.

D$1\frac{1}{2}$. [Пуста ли ссылка `LLINK`?] Если `LLINK(T)` $= \Lambda$, установить `Q` $\leftarrow$ `RLINK(T)` и перейти к шагу D4.

В упр. 14 показано, что алгоритм D с этим дополнительным шагом всегда оставляет дерево с той же, если не меньшей, длиной пути, что и алгоритм D; иногда же результат оказывается даже лучше ожидаемого. При комбинировании рассмотренного метода со стратегией симметричного удаления Эппингера длина пути при повторяющихся случайных удалениях/вставках уменьшается до примерно 86% от значения, получаемого при использовании вставок без удалений.

Частота обращений. Пока что мы предполагали, что каждый ключ может стать объектом поиска с одинаковой вероятностью. Однако в более общем случае следует рассмотреть вероятности p_k поиска k-го вставленного элемента ($p_1 + \cdots + p_N = 1$). Модифицированное выражение (2) (в предположении о случайном порядке дерево остается случайным и выполняется соотношение (5)) показывает, что среднее коли-

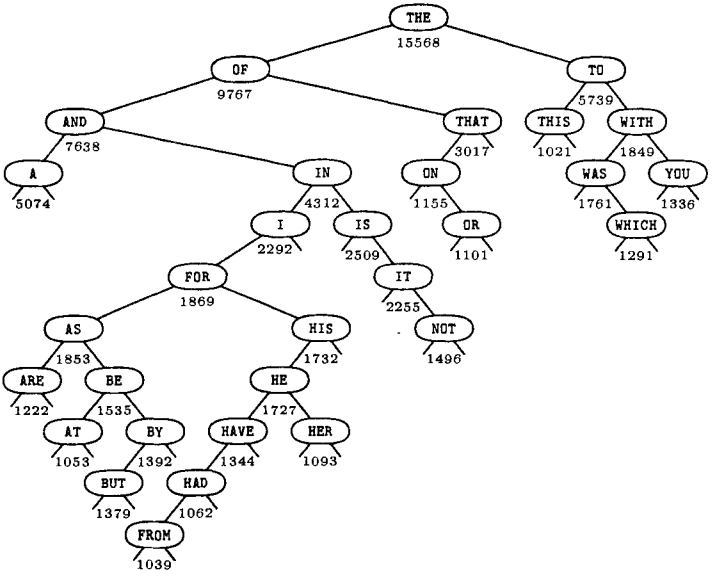


Рис. 12. 31 наиболее употребительное английское слово, вставленное в порядке уменьшения частоты.

чество сравнений в случае успешного поиска составляет

$$1 + \sum_{k=1}^{N} p_k(2H_k - 2) = 2\sum_{k=1}^{N} p_k H_k - 1. \tag{11}$$

Например, если вероятности соответствуют закону Зипфа (6.1–(8)), среднее количество сравнений сводится к

$$H_N - 1 + H_N^{(2)}/H_N \tag{12}$$

в предположении, что ключи будут вставляться в порядке уменьшения их важности (см. упр. 18). Эта величина примерно в два раза меньше величины, предсказываемой в результате анализа случая с равными частотами, и меньше, чем в случае бинарного поиска.

На рис. 12 показано дерево, полученное в результате вставки 31 наиболее часто употребляющегося английского слова в порядке убывания частоты появления в текстах. Относительные частоты, приведенные для каждого слова, взяты из H. F. Gaines, *Cryptanalysis* (New York: Dover, 1956), 226. Среднее количество сравнений при успешном поиске в этом дереве равно 4.042; соответствующее значение для бинарного поиска с использованием алгоритма 6.2.1B или 6.2.1C требует 4.393 сравнений.

Оптимальные бинарные деревья поиска. Рассмотренный материал заставляет задуматься о том, какое дерево является наилучшим для поиска в таблице ключей с заданными частотами. Например, для случая с 31 наиболее употребительным словом оптимальное дерево показано на рис. 13; оно требует в среднем всего лишь 3.437 сравнений при успешном поиске.

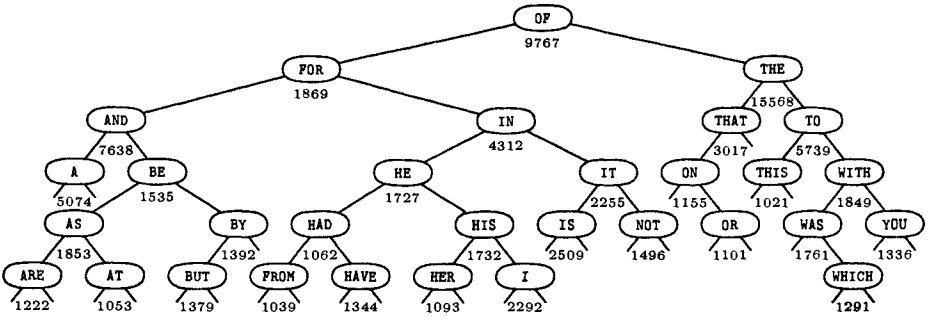
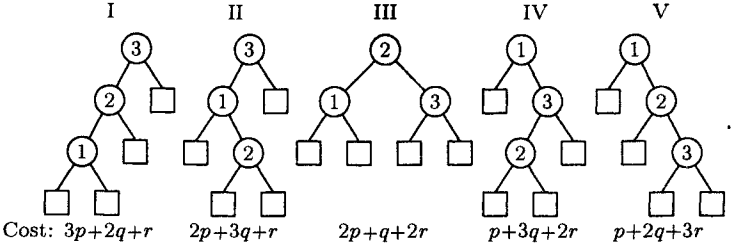


Рис. 13. Оптимальное дерево поиска для 31 наиболее употребительного английского слова.

Приступим к задаче поиска оптимального дерева. Например, при $N = 3$ в предположении, что ключи $K_1 < K_2 < K_3$ имеют вероятности p, q, r соответственно, возможны пять деревьев:



(13)

На рис. 14 показаны диапазоны значений p, q, r, для которых каждое из деревьев является оптимальным; сбалансированное дерево является наилучшим примерно в 45% случаев при случайном выборе p, q, r (см. упр. 21).

К сожалению, при больших N имеется

$$\binom{2N}{N} \Big/ (N+1) \approx 4^N \big/ \left(\sqrt{\pi}\, N^{3/2}\right)$$

бинарных деревьев, так что перебрать их все и найти наилучшие — совершенно неразрешимая задача. Поэтому сто́ит познакомиться с ними поближе, чтобы, лучше узнав свойства бинарных деревьев, найти среди них оптимальные.

До сих пор нас интересовал успешный поиск, однако на практике неудачный поиск не менее важен (более того, в некоторых специфичных задачах именно он является определяющим; случаями успешного поиска в таких задачах можно попросту пренебречь. — *Прим. перев.*). Например, представленные на рис. 13 слова соответствуют примерно 36% слов обычного английского текста; остальные 64%, естественно, повлияют на структуру оптимального дерева поиска.

Сформулируем задачу следующим образом. Даны $2n + 1$ вероятностей $p_1, p_2, \ldots, p_n$ и $q_0, q_1, \ldots, q_n$, где

p_i — вероятность того, что аргументом поиска является K_i;

q_i — вероятность того, что аргумент поиска лежит между K_i и K_{i+1}.

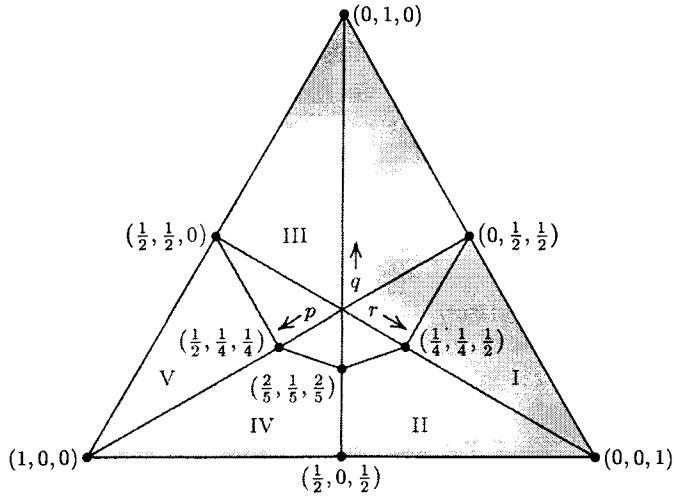
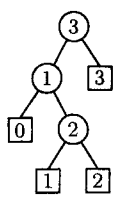


Рис. 14. На этой диаграмме показано, какое из пяти деревьев (13) является наилучшим в случае относительных частот (p, q, r) ключей (K_1, K_2, K_3). Тот факт, что $p + q + r = 1$, делает диаграмму двумерной несмотря на наличие трех координат.

(Пусть q_0 представляет собой вероятность того, что аргумент поиска меньше, чем K_1, а q_n — вероятность того, что аргумент поиска больше, чем K_n.) Таким образом, $p_1 + p_2 + \cdots + p_n + q_0 + q_1 + \cdots + q_n = 1$ и мы хотим найти бинарное дерево, минимизирующее ожидаемое количество сравнений при поиске, а именно

$$\sum_{j=1}^{n} p_j \big(\text{уровень}(\textcircled{j}) + 1\big) + \sum_{k=0}^{n} q_k \ \text{уровень}(\boxed{k}), \tag{14}$$

где $\textcircled{j}$ — j-й внутренний узел при симметричном обходе, а $\boxed{k}$ — $(k+1)$-й внешний узел; корень дерева находится на нулевом уровне. Так, ожидаемое количество сравнений для бинарного дерева



(15)

равно $2q_0 + 2p_1 + 3q_1 + 3p_2 + 3q_2 + p_3 + q_3$. Назовем это значение *ценой* дерева, а дерево с минимальной ценой — *оптимальным*. При таком определении требование, чтобы сумма всех p и q была равна 1, излишне — мы можем искать дерево с минимальной ценой с любой данной последовательностью весов $(p_1, \ldots, p_n;\ q_0, \ldots, q_n)$.

В разделе 2.3.4.5 мы изучали процедуру Хаффмана (Huffman) для построения деревьев с минимальной взвешенной длиной пути, однако этот метод требовал, чтобы все p были нулевыми, и, кроме того, в построенном этим методом дереве внешние узлы с весами $(q_0, \ldots, q_n)$ обычно *не* располагались в правильном порядке с точки зрения симметричного обхода. Следовательно, нам надо поискать другие подходы к решению этой задачи.

Нас спасет то, что все *поддеревья оптимального дерева оптимальны*. Например, если (15) представляет собой оптимальное дерево для весов $(p_1, p_2, p_3;\ q_0, q_1, q_2, q_3)$, то его левое поддерево должно быть оптимально для $(p_1, p_2;\ q_0, q_1, q_2)$; любое улучшение поддерева должно приводить к улучшению дерева в целом.

Данный принцип предлагает вычислительную процедуру, которая систематично ищет все бóльшие и бóльшие оптимальные поддеревья. Мы использовали эту же идею в разделе 5.4.9 для построения схем оптимального слияния; общий подход, известный как "динамическое программирование", будет рассмотрен в разделе 7.7*.

Пусть $c(i,j)$ — цена оптимального поддерева с весами $(p_{i+1}, \ldots, p_j;\ q_i, \ldots, q_j)$ и пусть $w(i,j) = p_{i+1} + \cdots + p_j + q_i + \cdots + q_j$ — сумма этих весов (очевидно, что $c(i,j)$ и $w(i,j)$ определены для $0 \le i \le j \le n$). Поскольку минимально возможная цена дерева с корнем k равна $w(i,j) + c(i,k-1) + c(k,j)$, получим

$$\begin{aligned} c(i,i) &= 0, \\ c(i,j) &= w(i,j) + \min_{i<k\le j}\big(c(i,k-1) + c(k,j)\big) \qquad \text{при } i<j. \end{aligned} \tag{16}$$

Через $R(i,j)$ при $i < j$ обозначим множество всех k, для которых достигается минимум в (16); это множество определяет корни оптимальных поддеревьев.

С помощью уравнений (16) можно вычислить $c(i,j)$ для $j-i = 1, 2, \ldots, n$; имеется примерно $\frac{1}{2}n^2$ таких значений, а операция минимизации выполняется примерно для $\frac{1}{6}n^3$ значений k. Это означает, что мы в состоянии определить оптимальное дерево за $O(n^3)$ единиц времени с использованием $O(n^2)$ ячеек памяти.

При оценке времени работы свойство монотонности позволяет уменьшить степень при n. Пусть $r(i,j)$ означает элемент множества $R(i,j)$; нет необходимости в вычислении всего множества $R(i,j)$ — достаточно одного его представителя. Найдя $r(i,j-1)$ и $r(i+1,j)$ и используя результат упр. 27, мы всегда можем полагать, что

$$r(i,j-1) \le r(i,j) \le r(i+1,j) \tag{17}$$

при неотрицательных весах. Это ограничивает поиск минимума всего лишь $r(i+1, j) - r(i,j-1) + 1$ проверяемыми в (16) значениями k вместо $j-i$. Общий объем работы при $j-i = d$ оценивается как

$$\sum_{\substack{d\le j\le n\\ i=j-d}} \big(r(i+1,j) - r(i,j-1) + 1\big) = r(n-d+1,n) - r(0,d-1) + n - d + 1 < 2n,$$

так что общее время работы уменьшается до $O(n^2)$.

Детально эта процедура рассматривается в описании алгоритма K.

Алгоритм K (*Поиск оптимальных бинарных деревьев поиска*). Даны $2n+1$ неотрицательных веса $(p_1, \ldots, p_n;\ q_0, \ldots, q_n)$. Алгоритм строит бинарные деревья $t(i,j)$, имеющие минимальную цену (в определенном ранее смысле) для весов $(p_{i+1}, \ldots, p_j;\ q_i, \ldots, q_j)$. Вычисляются три массива, а именно

$c[i,j]$,	$0 \le i \le j \le n$,	цена $t(i,j)$;
$r[i,j]$,	$0 \le i < j \le n$,	корень $t(i,j)$;
$w[i,j]$,	$0 \le i \le j \le n$,	общий вес $t(i,j)$.

---

* Имеется в виду раздел из планируемого автором тома 4. — *Прим. перев.*

Результаты работы алгоритма определяются массивом r: при $i = j$ $t(i,j)$ пуст; в противном случае левое поддерево — $t(i, r[i,j]-1)$, а правое поддерево — $t(r[i,j], j)$.

K1. [Инициализация.] Для $0 \le i \le n$ установить $c[i,i] \leftarrow 0$, $w[i,i] \leftarrow q_i$ и $w[i,j] \leftarrow w[i,j-1]+p_j+q_j$ при $j = i+1, \ldots, n$. Затем для $1 \le j \le n$ установить $c[j-1,j] \leftarrow w[j-1,j]$ и $r[j-1,j] \leftarrow j$. (Эти действия определяют все оптимальные деревья из одного узла.)

K2. [Цикл по d.] Выполнить шаг K3 для $d = 2, 3, \ldots, n$, затем остановить работу алгоритма.

K3. [Цикл по j.] (К этому моменту уже определены оптимальные деревья с менее чем d узлами. На данном шаге определяются все оптимальные деревья с d узлами). Выполнить шаг K4 для $j = d, d+1, \ldots, n$.

K4. [Поиск $c[i,j]$, $r[i,j]$.] Установить $i \leftarrow j - d$, затем —

$$c[i,j] \leftarrow w[i,j] + \min_{r[i,j-1] \le k \le r[i+1,j]} \left(c[i,k-1] + c[k,j]\right)$$

и присвоить $r[i,j]$ значение k, для которого достигается минимум. (В упр. 22 доказывается, что $r[i,j-1] \le r[i+1,j]$.) ▮

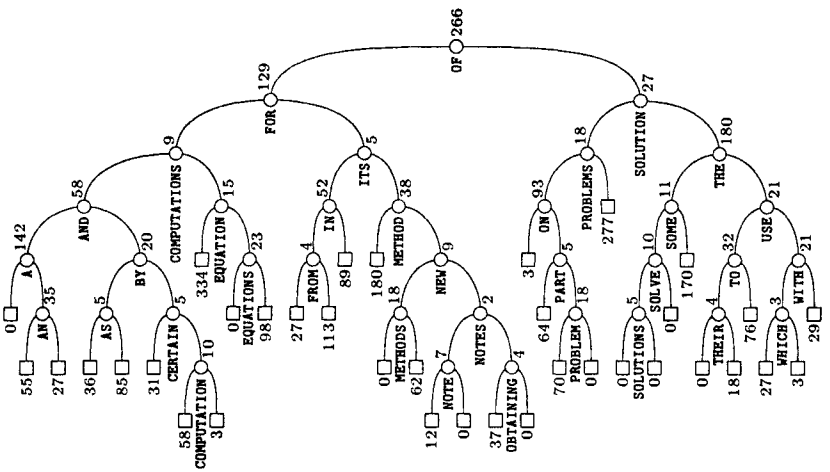


Рис. 15. Оптимальное бинарное дерево поиска для указателя KWIC.

В качестве примера алгоритма K рассмотрим рис. 15, основанный на приложении индексирования "key-word-in-context" (ключевое слово в контексте — KWIC). Заглавия всех статей в первых десяти томах *Journal of the ACM* были рассортированы таким образом, чтобы каждому слову каждого заголовка соответствовала одна строка. Впрочем, некоторые слова наподобие THE и EQUATION были признаны неинформативными и не вошли в указатель. Эти специальные слова и частота их появления обозначены внутренними узлами на рис. 15. Обратите внимание на то, что заглавие наподобие "On the solution of an equation for a certain new problem" ("О решении уравнения для некоторой новой задачи") оказывается настолько неинформативным, что вовсе не попадает в указатель! Идея указателя KWIC описана

в работе H. P. Luhn, *Amer. Documentation* **11** (1960), 288–295, а сам указатель полностью приведен в работе W. W. Youden, *JACM* **10** (1963), 583–646.

При подготовке указателя KWIC к сортировке нам, возможно, понадобилось бы воспользоваться бинарным деревом поиска, чтобы проверить, должно ли определенное слово быть внесенным в указатель. Другие слова, попадающие в алфавитном порядке между неиндексируемыми словами, показаны в виде внешних узлов на рис. 15. Так, в заглавиях статей *JACM* за 1954–1963 годы встретилось ровно 277 слов, расположенных по алфавиту между словами PROBLEMS и SOLUTION.

На рис. 15 показано оптимальное дерево, полученное согласно алгоритму K при $n = 35$. Вычисленные для $j = 1, 2, \ldots, 35$ значения $r[0, j]$ равны $(1, 1, 2, 3, 3, 3, 3, 8, 8, 8, 8, 8, 8, 11, 11, \ldots, 11, 21, 21, 21, 21, 21, 21)$; значения $r[i, 35]$ для $i = 0, 1, \ldots, 34$ равны $(21, 21, \ldots, 21, 25, 25, 25, 25, 25, 25, 26, 26, 26, 30, 30, 30, 30, 30, 30, 30, 33, 33, 33, 35, 35)$.

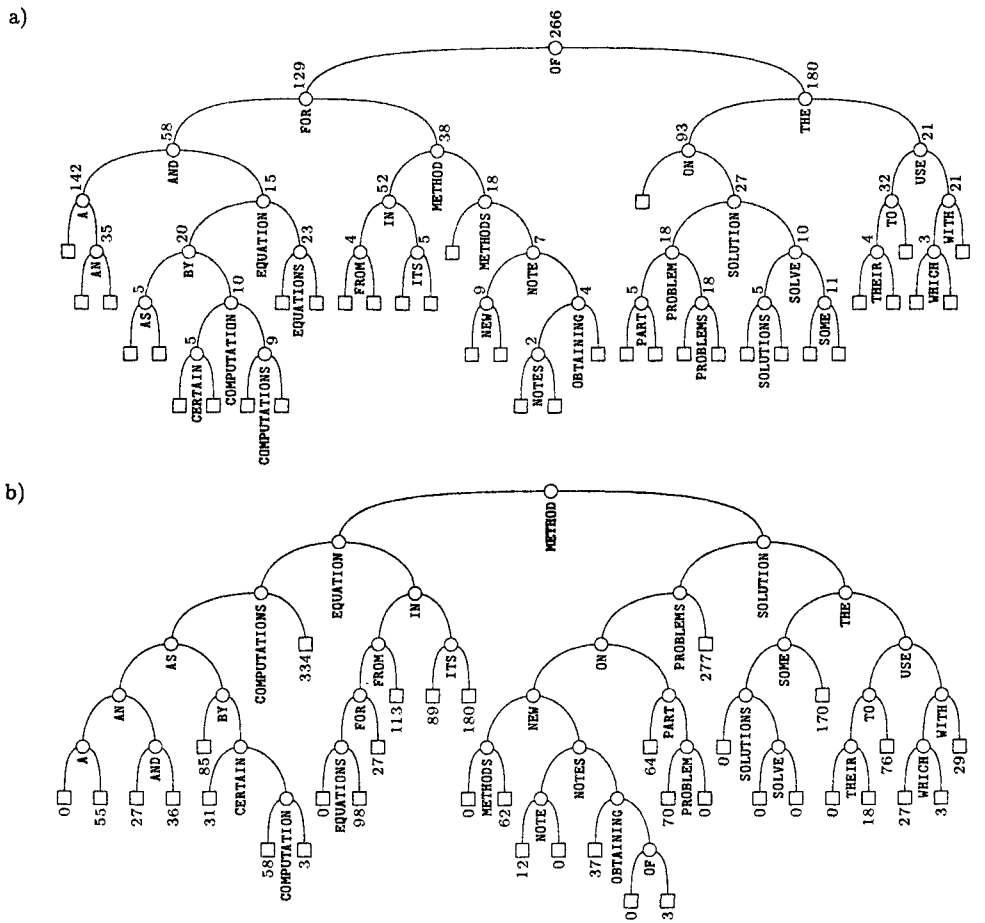


Рис. 16. Оптимальные бинарные деревья поиска, основанные на половине данных, которые представлены на рис. 15: (a) без учета внешних частот; (b) без учета внутренних частот.

"Промежуточные частоты" q_j оказывают существенное влияние на структуру оптимального дерева; на рис. 16, (a) показано оптимальное дерево, которое было бы получено при всех q_j, равных нулю. Точно так же важны и внутренние частоты p_i; на рис. 16, (b) представлено оптимальное дерево при p_i, равных нулю. При полном наборе частот дерево, изображенное на рис. 15, требует в среднем только 4.15 сравнений; деревья на рис. 16 требуют 4.69 и 4.55 сравнений соответственно.

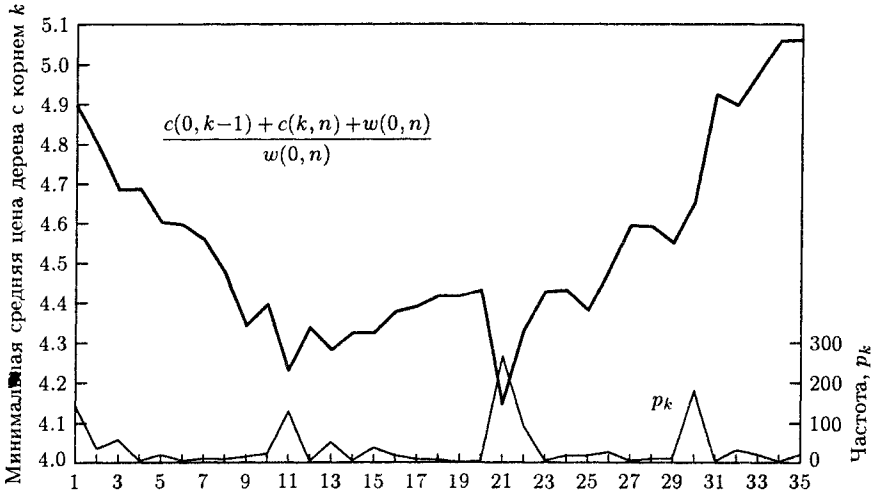


Рис. 17. Поведение цены как функции корня k.

Поскольку для алгоритма K необходимы время и пространство, пропорциональные n^2, его использование непрактично при больших n. Конечно, при очень больших n мы можем отказаться от использования бинарных деревьев и подумать о других методах поиска, которые будут рассмотрены нами немного позже. А пока предположим, что надо найти оптимальное (или почти оптимальное) дерево при больших значениях n.

Мы видели, что идея вставки ключей в порядке уменьшения частот приводит к получению в среднем неплохих деревьев; впрочем, они могут оказаться и очень плохими (см. упр. 20), так как веса q_j при таком методе построения не используются. Еще один подход состоит в выборе корня k таким образом, чтобы максимальный вес поддерева $\max\bigl(w(0,k-1),\ w(k,n)\bigr)$ был настолько мал, насколько это возможно. Такой подход также может оказаться плохим (при выборе в качестве корня узла с малым значением p_k). Впрочем, теорема M, приведенная ниже, показывает, что полученное в результате дерево будет не так уж далеко от оптимального.

Более успешная процедура может быть получена путем комбинирования описанных методов, как было предложено В. А. Волкером (W. A. Walker) и К. К. Готлибом (C. C. Gotlieb) (*Graph Theory and Computing* (Academic Press, 1972), 303–323). Попытайтесь уравнять "левые" и "правые" веса, однако будьте готовы переместить корень на несколько шагов вправо или влево для поиска узла с относительно бóльшим p_k. На рис. 17 показана "разумность" предложенного метода: если начер-

тить график $c(0,k-1)+c(k,n)$ как функции от k для данных KWIC (см. рис. 15), то увидим, что результат весьма чувствителен к величине p_k.

Такой метод "сверху вниз" можно использовать при больших n для выбора корня и дальнейшей работы с левыми и правыми поддеревьями. При получении достаточно малого поддерева возможно применение алгоритма K. Результирующий метод дает неплохие деревья (по сообщениям отличающиеся от оптимальных на 2–3%), требуя при этом пространство $O(n)$ и время работы — $O(n\log n)$ единиц. М. Фредманом (M. Fredman) было показано, что на самом деле достаточно $O(n)$ единиц времени при использовании подходящих структур данных (*STOC* **7** (1975), 240–244; см. также K. Mehlhorn , *Data Structures and Algorithms* **1** (Springer, 1984), Section 4.2).

Оптимальные деревья и энтропия. Минимальная цена очень близка к другому математическому понятию, именуемому *энтропией*, которое было введено Клодом Шенноном (Claude Shannon) в его знаменитой работе по теории информации (*Bell System Tech. J.* **27** (1948), 379–423, 623–656). Если p_1, p_2, ..., p_n представляют собой вероятности, причем $p_1+p_2+\cdots+p_n=1$, мы определяем энтропию $H(p_1,p_2,\ldots,p_n)$ как

$$H(p_1,p_2,\ldots,p_n) = \sum_{k=1}^{n} p_k \lg\frac{1}{p_k}. \tag{18}$$

Интуитивно понятно, что если происходит k-е событие (с вероятностью p_k) из n возможных, мы получаем $\lg(1/p_k)$ бит информации (так, событие, вероятность которого равна $\frac{1}{32}$, дает нам 5 бит информации). Значит, $H(p_1,p_2,\ldots,p_n)$ представляет собой ожидаемое число битов информации при наступлении случайного события. Если $p_k=0$, то определим $p_k\lg(1/p_k)=0$, поскольку $\lim_{\epsilon\to0+}\epsilon\lg\frac{1}{\epsilon}=\lim_{m\to\infty}\frac{1}{m}\lg m=0$. Такое соглашение позволяет нам пользоваться формулой (18) даже в случае, когда некоторые вероятности равны нулю.

Функция $x\lg(1/x)$ вогнута, т. е. ее вторая производная, $-1/(x\ln2)$, отрицательна*. Таким образом, при $p_1=p_2=\cdots=p_n=1/n$ достигается максимальное значение энтропии $H(p_1,p_2,\ldots,p_n)$, равное

$$H\left(\frac{1}{n},\frac{1}{n},\ldots,\frac{1}{n}\right)=\lg n. \tag{19}$$

В общем случае, когда мы фиксируем значения p_1, ..., p_{n-k} и позволяем изменяться значениям p_{n-k+1}, ..., p_n, справедливы неравенства

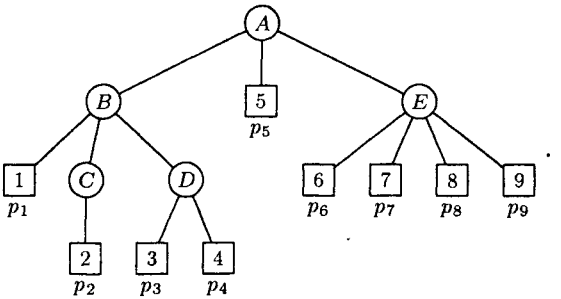
$$\begin{aligned} H(p_1,\ldots,p_{n-k},p_{n-k+1},\ldots,p_n) &\le H\left(p_1,\ldots,p_{n-k},\frac{q}{k},\ldots,\frac{q}{k}\right)\\ &= H(p_1,\ldots,p_{n-k},q)+q\lg k, \end{aligned} \tag{20}$$

$$\begin{aligned} H(p_1,\ldots,p_{n-k},p_{n-k+1},\ldots,p_n) &\ge H(p_1,\ldots,p_{n-k},q,0,\ldots,0)\\ &= H(p_1,\ldots,p_{n-k},q), \end{aligned} \tag{21}$$

где $q=1-(p_1+\cdots+p_{n-k})$.

* Автор использует для описания этой функции термин *concave*, дословно переводимый как *вогнутый*. Следует отметить, что в отечественной математике зачастую используются более точные термины *выпуклая вниз* и *выпуклая вверх* функции (Бронштейн И. Н., Семендяев К. А. *Справочник по математике для инженеров и учащихся втузов*. — М.: Наука, 1981. — 720 с.). Здесь и далее под *вогнутой* функцией подразумевается *выпуклая вверх* функция. — *Прим. перев.*

Рассмотрим любое (не обязательно бинарное) дерево, листьям которого приписаны определенные вероятности, например



(22)

Здесь p_k — вероятности того, что поисковая процедура завершит работу в листе $\boxed{k}$. Ветвление в каждом внутреннем узле соответствует локальному распределению вероятностей (основанному на сумме вероятностей листьев каждой ветви). Например, первая, вторая и третья ветви узла (A) имеют соответственно вероятности $(p_1 + p_2 + p_3 + p_4,\ p_5,\ p_6 + p_7 + p_8 + p_9)$; вероятности же в узле (B) равны $(p_1, p_2, p_3 + p_4)/(p_1 + p_2 + p_3 + p_4)$. Мы можем утверждать, что каждый внутренний узел имеет энтропию своего локального распределения вероятностей; таким образом,

$$
\begin{aligned}
H(A) &= (p_1+p_2+p_3+p_4)\lg\frac{1}{p_1+p_2+p_3+p_4} \\
&\quad + p_5\lg\frac{1}{p_5} + (p_6+p_7+p_8+p_9)\lg\frac{1}{p_6+p_7+p_8+p_9}, \\
H(B) &= \frac{p_1}{p_1+p_2+p_3+p_4}\lg\frac{p_1+p_2+p_3+p_4}{p_1} + \frac{p_2}{p_1+p_2+p_3+p_4}\lg\frac{p_1+p_2+p_3+p_4}{p_2} \\
&\quad + \frac{p_3+p_4}{p_1+p_2+p_3+p_4}\lg\frac{p_1+p_2+p_3+p_4}{p_3+p_4}, \\
H(C) &= \frac{p_2}{p_2}\lg\frac{p_2}{p_2}, \\
H(D) &= \frac{p_3}{p_3+p_4}\lg\frac{p_3+p_4}{p_3} + \frac{p_4}{p_3+p_4}\lg\frac{p_3+p_4}{p_4}, \\
H(E) &= \frac{p_6}{p_6+p_7+p_8+p_9}\lg\frac{p_6+p_7+p_8+p_9}{p_6} + \frac{p_7}{p_6+p_7+p_8+p_9}\lg\frac{p_6+p_7+p_8+p_9}{p_7} \\
&\quad + \frac{p_8}{p_6+p_7+p_8+p_9}\lg\frac{p_6+p_7+p_8+p_9}{p_8} + \frac{p_9}{p_6+p_7+p_8+p_9}\lg\frac{p_6+p_7+p_8+p_9}{p_9}.
\end{aligned}
$$

Лемма E. *Сумма $p(\alpha)H(\alpha)$ по всем внутренним узлам дерева α, где $p(\alpha)$ — вероятность достижения узла α, а $H(\alpha)$ — энтропия α, равна энтропии распределения вероятностей листьев.*

Доказательство. Это утверждение легко доказывается по индукции снизу вверх. Например, в соответствии с приведенными выше формулами имеем:

$$H(A)+(p_1+p_2+p_3+p_4)H(B)+p_2H(C)+(p_3+p_4)H(D)+(p_6+p_7+p_8+p_9)H(E)$$
$$=p_1\lg\frac{1}{p_1}+p_2\lg\frac{1}{p_2}+\cdots+p_9\lg\frac{1}{p_9}$$

(все множители, включающие $\lg(p_1+p_2+p_3+p_4)$, $\lg(p_3+p_4)$ и $\lg(p_6+p_7+p_8+p_9)$, сокращаются). ▌

На основании леммы E мы можем использовать энтропию для определения нижней границы цены любого бинарного дерева.

Теорема B. *Пусть* $(p_1,\ldots,p_n;q_0,\ldots,q_n)$ *— неотрицательные веса из алгоритма K, нормализованные таким образом, что* $p_1+\cdots+p_n+q_0+\cdots+q_n=1$, *а* $P=p_1+\cdots+p_n$ *— вероятность успешного поиска. Пусть также*

$$H=H(p_1,\ldots,p_n,q_0,\ldots,q_n)$$

представляет собой энтропию соответствующего распределения вероятностей, а C *— минимальную цену* (14). *Тогда при* $H\ge 2P/e$ *имеем*

$$C\ \ge\ H-P\lg\frac{eH}{2P}. \tag{23}$$

Доказательство. Возьмем бинарное дерево стоимости C и назначим вероятности q_k его листьям. Добавим к каждому внутреннему узлу среднюю ветвь, на которой будет размещен лист, имеющий вероятность p_k. Тогда $C=\sum p(\alpha)$, где суммирование производится по всем внутренним узлам α полученного тернарного дерева и согласно лемме E $H=\sum p(\alpha)H(\alpha)$.

Энтропия $H(\alpha)$ соответствует распределению вероятности для трех событий, где одна из вероятностей (в случае, когда α — внутренний узел $\textcircled{j}$) равна $p_j/p(\alpha)$. В упр. 35 доказывается, что

$$H(p,q,r)\le p\lg x+1+\lg\Bigl(1+\frac{1}{2x}\Bigr) \tag{24}$$

для всех $x>0$, когда $p+q+r=1$. Следовательно, для всех положительных x выполняется неравенство

$$H=\sum_\alpha p(\alpha)H(\alpha)\ \le\ \sum_{j=1}^n p_j\lg x\ +\ \Bigl(1+\lg\Bigl(1+\frac{1}{2x}\Bigr)\Bigr)C.$$

Выполнив подстановку $2x=H/P$, получаем требуемый результат, поскольку

$$\begin{aligned}C&\ge\frac{1}{1+\lg(1+P/H)}\Bigl(H-P\lg\frac{H}{2P}\Bigr)\\&=\frac{1}{1+\lg(1+P/H)}(H+P\lg e)-\frac{P}{1+\lg(1+P/H)}\lg\frac{eH}{2P}\\&\ge H-P\lg\frac{eH}{2P},\end{aligned}$$

так как $\lg(1+y)\le y\lg e$ для всех $y>0$. ▌

Неравенство (23) может не выполняться для крайне малых значений энтропии; ограничение случаем $H \geq 2P/e$ не слишком строго, поскольку обычно значение энтропии составляет порядка $\lg n$ (см. упр. 37). Обратите также внимание на то, что в доказательстве нигде не использовался порядок узлов "слева направо"; таким образом, нижняя граница (23) справедлива для всех бинарных деревьев поиска с вероятностями внутренних узлов p_j и внешних q_k в любом порядке.

Вычисления энтропии дают нам также верхнюю границу, которая не сильно отличается от (23), даже если мы будем придерживаться порядка "слева направо".

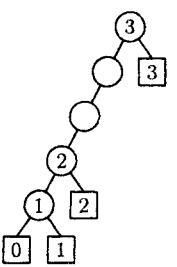
Теорема M. *В предположениях теоремы B справедливо также неравенство*

$$C < H + 2 - P. \tag{25}$$

Доказательство. Построим $n+1$ сумму $s_0 = \frac{1}{2}q_0$, $s_1 = q_0 + p_1 + \frac{1}{2}q_1$, $s_2 = q_0 + p_1 + q_1 + p_2 + \frac{1}{2}q_2$, $\ldots$, $s_n = q_0 + p_1 + \cdots + q_{n-1} + p_n + \frac{1}{2}q_n$; согласно упр. 38 мы можем считать, что $s_0 < s_1 < \cdots < s_n$. Записывая каждую s_k как двоичную дробь, получим $s_n = (.111\ldots)_2$ для $s_n = 1$. Пусть строка σ_k представляет собой начальные биты s_k, необходимые для того, чтобы отличать s_k от s_j при $j \neq k$. Например, для $n = 3$ и приведенных значений s_k

$$\begin{aligned} s_0 &= (.0000001)_2 & \sigma_0 &= 00000 \\ s_1 &= (.0000101)_2 & \sigma_1 &= 00001 \\ s_2 &= (.0001011)_2 & \sigma_2 &= 0001 \\ s_3 &= (.1100000)_2 & \sigma_3 &= 1 \end{aligned}$$

Построим бинарное дерево с $n+1$ листом таким образом, чтобы σ_k соответствовало пути от корня к $\boxed{k}$ для $0 \leq k \leq n$, где "0" означает левую, а "1" — правую ветви. Кроме того, если σ_{k-1} имеет вид $\alpha_k 0 \beta_k$, а σ_k — $\alpha_k 1 \gamma_k$ для некоторых α_k, β_k и γ_k, то внутренний узел $\textcircled{k}$ соответствует пути α_k. Таким образом, дерево



соответствует приведенному ранее примеру. Как видите, у дерева могут остаться безымянные внутренние узлы; заменим каждый из них его единственным дочерним узлом. Цена полученного дерева не будет превышать $\sum_{k=1}^{n} p_k(|\alpha_k|+1) + \sum_{k=0}^{n} q_k|\sigma_k|$.

Поскольку $s_k \leq (.\alpha_k)_2 + 2^{-|\alpha_k|}$ и $s_{k-1} \geq (.\alpha_k)_2$, получим

$$p_k \leq \tfrac{1}{2}q_{k-1} + p_k + \tfrac{1}{2}q_k = s_k - s_{k-1} \leq 2^{-|\alpha_k|}. \tag{26}$$

Более того, в случае $q_k \geq 2^{-t}$ имеем $s_k \geq s_{k-1} + 2^{-t-1}$ и $s_{k+1} \geq s_k + 2^{-t-1}$; таким образом, $|\sigma_k| \leq t+1$. Отсюда следует, что $q_k < 2^{-|\sigma_k|+2}$ и мы построили бинарное дерево ценой

$$\le \sum_{k=1}^{n} p_k(1+|\alpha_k|) + \sum_{k=0}^{n} q_k|\sigma_k| \le \sum_{k=1}^{n} p_k\left(1+\lg\frac{1}{p_k}\right) + \sum_{k=0}^{n} q_k\left(2+\lg\frac{1}{q_k}\right)$$
$$= P + 2(1-P) + H = H + 2 - P. \quad \blacksquare$$

В случае указателя KWIC (см. рис. 15) $P = 1304/3288 \approx 0.39659$ и $H(p_1, \ldots, p_{35}, q_0, \ldots, q_{35}) \approx 5.00635$. Вследствие этого теорема B утверждает, что $C \ge 3.3800$ и согласно теореме M $C < 6.6098$.

***Алгоритм Гарсия-Воча.** Возможно поразительное улучшение алгоритма K в специальном случае: $p_1 = \cdots = p_n = 0$. Этот случай, когда роль играют только вероятности листьев $(q_0, q_1, \ldots, q_n)$, особенно важен, так как он нередко встречается во многих приложениях. Далее в этом разделе мы предполагаем, что вероятности p_j — нулевые. В данном случае теоремы B и M сводятся к неравенствам

$$H(q_0, q_1, \ldots, q_n) \le C(q_0, q_1, \ldots, q_n) < H(q_0, q_1, \ldots, q_n) + 2 \tag{27}$$

и функция цены (14) упрощается до

$$C = \sum_{k=0}^{n} q_k l_k, \qquad l_k = \text{уровень } \boxed{k}. \tag{28}$$

Ключевое свойство, позволяющее упростить алгоритм, заключается в следующем наблюдении.

Лемма W. *Если $q_{k-1} > q_{k+1}$, то в каждом оптимальном дереве $l_k \le l_{k+1}$. Если же $q_{k-1} = q_{k+1}$, то в некоторых оптимальных деревьях $l_k \le l_{k+1}$.*

Доказательство. Предположим, что $q_{k-1} \ge q_{k+1}$, и рассмотрим дерево, в котором $l_k > l_{k+1}$. Тогда $\boxed{k}$ должен быть правым дочерним узлом, а его левый "брат" L — поддеревом веса $c \ge q_{k-1}$. Заменим родительский по отношению к $\boxed{k}$ узел поддеревом L, а узел $\boxed{k+1}$ — узлом, дочерними узлами которого являются $\boxed{k}$ и $\boxed{k+1}$. Такая замена изменит общую стоимость на $-c - q_k(l_k - l_{k+1} - 1) + q_{k+1} \le q_{k+1} - q_{k-1}$. Значит, данное дерево не было оптимальным, если выполнялось условие $q_{k-1} > q_{k+1}$; в случае $q_{k-1} = q_{k+1}$ оптимальное дерево было трансформировано в другое оптимальное дерево. В последнем случае последовательность таких трансформаций приведет к $l_k \le l_{k+1}$. $\blacksquare$

Более глубокий анализ структуры даст нам значительно больше.

Лемма X. *Предположим, что j и k — индексы, такие, что $j < k$ и*

i) $q_{i-1} > q_{i+1}$ *при* $1 \le i < k$;
ii) $q_{k-1} \le q_{k+1}$;
iii) $q_i < q_{k-1} + q_k$ *при* $j \le i < k-1$;
iv) $q_{j-1} \ge q_{k-1} + q_k$.

Тогда существует оптимальное дерево, в котором $l_{k-1} = l_k$ и либо

a) $l_j = l_k - 1$, *либо*
b) $l_j = l_k$ *и* $\boxed{j}$ *является левым дочерним узлом.*

Доказательство. При замене "левого" "правым" в лемме W мы увидим, что (ii) означает существование оптимального дерева, в котором $l_{k-1} \geq l_k$. Однако лемма W и (i) означают, что $l_1 \leq l_2 \leq \cdots \leq l_k$. Следовательно, $l_{k-1} = l_k$.

Предположим, что $l_s < l_k - 1 \leq l_{s+1}$ для некоторого $j \leq s < k$. Пусть t представляет собой наименьший индекс, меньший k, такой, что $l_t = l_k$. Тогда $l_{s+1} = \cdots = l_{t-1} = l_k - 1$ и $\boxed{s+1}$ представляет собой левый дочерний узел. Отсюда вытекает, что $t-s$ нечетно и узел $\boxed{i}$ является левым дочерним узлом при $i = s+1, s+3, \ldots, t$. Заменим родительский узел узла $\boxed{t}$ узлом $\boxed{t+1}$, а узел $\boxed{i}$ — узлом $\boxed{i+1}$ для $s < i < t$. Кроме того, заменим внешний узел $\boxed{s}$ внутренним узлом, дочерними узлами которого являются $\boxed{s}$ и $\boxed{s+1}$. Все эти замены приведут к изменению цены на $\leq q_s - q_t - q_{t+1} \leq q_s - q_{k-1} - q_k$, т. е. к улучшению дерева при $q_s < q_{k-1} + q_k$. Следовательно, согласно (iii) $l_j \geq l_k - 1$.

До этого момента мы ни разу не использовали гипотезу (iv). Если $l_j = l_k$ и $\boxed{j}$ не является левым дочерним узлом, то $\boxed{j}$ должен быть правым "братом" узла $\boxed{j-1}$. Заменим их родительский узел узлом $\boxed{j-1}$, а затем заменим каждый лист $\boxed{i}$ листом $\boxed{i-1}$ для $j < i < k$. Заменим также внешний узел $\boxed{k}$ внутренним узлом, являющимся родительским по отношению к $\boxed{k-1}$ и $\boxed{k}$. Эти действия изменят цену на $-q_{j-1} + q_{k-1} + q_k \leq 0$, так что мы получим оптимальное дерево, удовлетворяющее условию (b). ∎

Лемма Y. *Пусть j и k те же, что и в лемме X. Рассмотрим измененные вероятности $(q'_0, \ldots, q'_{n-1}) = (q_0, \ldots, q_{j-1}, q_{k-1} + q_k, q_j, \ldots, q_{k-2}, q_{k+1}, \ldots, q_n)$, полученные посредством удаления q_{k-1} и q_k и вставки $q_{k-1} + q_k$ после q_{j-1}. Тогда*

$$C(q'_0, \ldots, q'_{n-1}) \;\leq\; (q_{k-1} + q_k) + C(q_0, \ldots, q_n). \tag{29}$$

Доказательство. Достаточно показать, что любое оптимальное дерево для $(q_0, \ldots, q_n)$ может быть трансформировано в дерево той же цены, в котором $\boxed{k-1}$ и $\boxed{k}$ являются "братьями", т. е. имеют один и тот же родительский узел, и листья которого появляются в следующем порядке:

$$\boxed{0} \quad \boxed{j-1} \quad \boxed{k-1} \quad \boxed{k} \quad \boxed{j} \quad \ldots \quad \boxed{k-2} \quad \boxed{k+1} \quad \ldots \quad \boxed{n}. \tag{30}$$

Мы начнем с дерева, сконструированного в лемме X. Если это дерево типа (b), просто переименуем листья, сместив $\boxed{k-1}$ и $\boxed{k}$ влево на $k-1-j$ мест. Если же это дерево типа (a), предположим, что $l_{s-1} = l_k - 1$ и $l_s = l_k$. В этом случае мы действуем следующим образом: сначала сместим $\boxed{k-1}$ и $\boxed{k}$ влево на $k-1-s$ мест, затем заменим их новый родительский узел узлом, дочерними узлами которого являются $\boxed{k-1}$ и $\boxed{k}$, а также заменим узел $\boxed{i}$ узлом $\boxed{i-1}$ для всех $j < i < s$. ∎

Лемма Z. *При выполнении условий леммы Y неравенство* (29) *становится равенством.*

Доказательство. Каждое дерево для $(q'_0, \ldots, q'_{n-1})$ соответствует дереву с листьями (30), в котором выпадающие из общего порядка листья $\boxed{k-1}$ и $\boxed{k}$ представляют дочерние узлы одного и того же родительского узла. Пусть этот родительский узел — (x). Мы хотим показать, что любое оптимальное дерево этого типа может быть конвертировано в дерево той же цены, но в котором листья располагаются в нормальном порядке — $\boxed{0} \ldots \boxed{n}$.

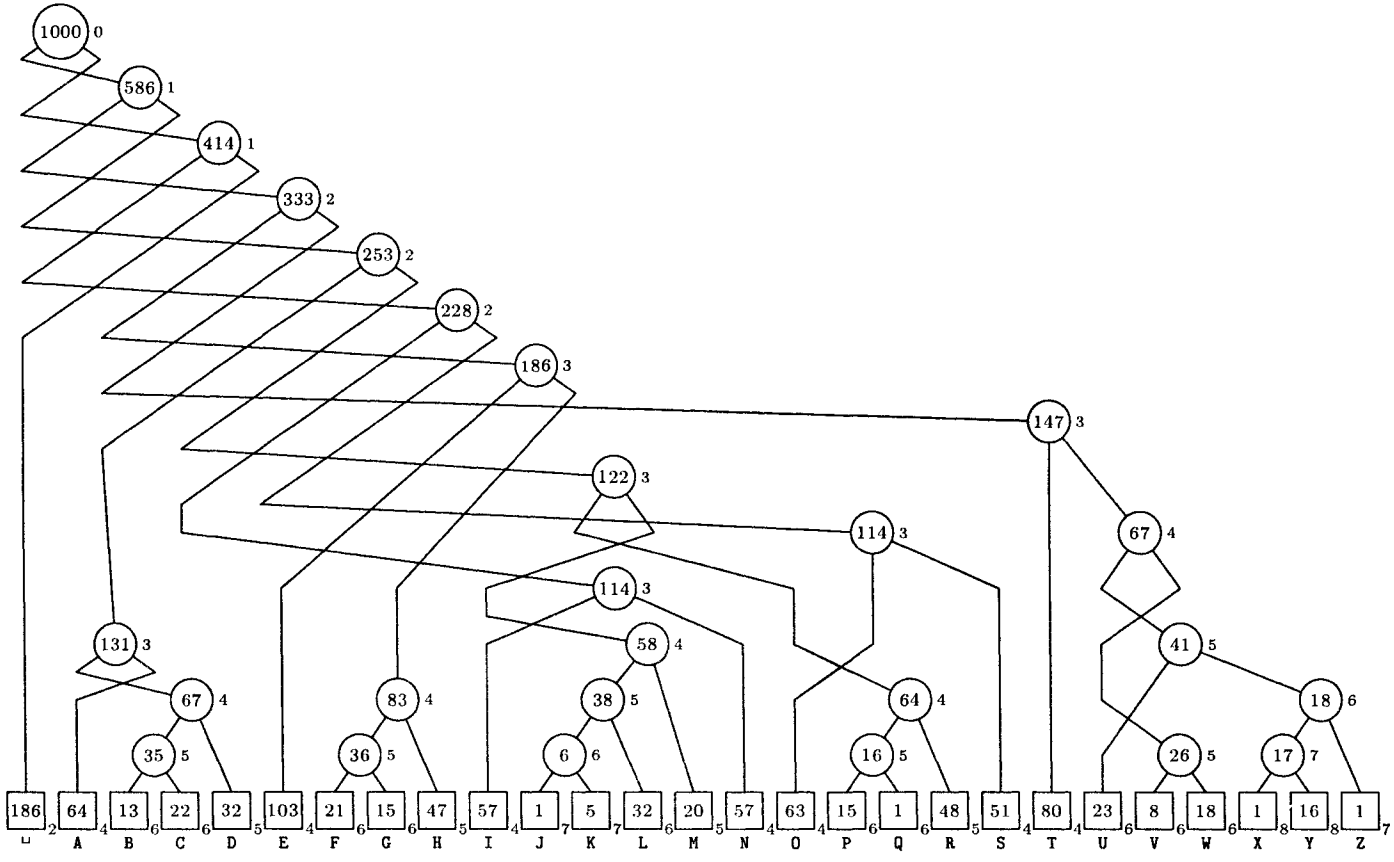


Рис. 18. Применение алгоритма Гарсия-Воча к данным о частотах появления букв; фазы 1 и 2.

Если $j = k-1$, доказывать нечего. В противном случае мы имеем $q'_{i-1} > q'_{i+1}$ для $j \le i < k-1$, поскольку $q_{j-1} \ge q_{k-1} + q_k > q_j$. Следовательно, согласно лемме W $l_x \le l_j \le \cdots \le l_{k-2}$, где l_x — уровень $\boxed{x}$ и l_i — уровень $\boxed{i}$ для $j \le i < k-1$. Если же $l_x = l_{k-2}$, мы просто смещаем узел $\boxed{x}$ вправо, замещая последовательность $\boxed{x}$ $\boxed{j}$ ... $\boxed{k-2}$ последовательностью $\boxed{j}$ ... $\boxed{k-2}$ $\boxed{x}$; это расположит листья в требуемом порядке.

В противном случае предположим, что $l_s = l_x$ и $l_{s+1} > l_x$. Сначала заменим $\boxed{x}$ $\boxed{j}$ ... $\boxed{s}$ на $\boxed{j}$ ... $\boxed{s}$ $\boxed{x}$; получим $l \le l_{s+1} \le \cdots \le l_{k-2}$, где $l = l_x + 1$ — общий уровень узлов $\boxed{k-1}$ и $\boxed{k}$. И, наконец, заменим узлы

$$\boxed{k-1}\ \boxed{k}\ \boxed{s+1}\ \ldots\ \boxed{k-2}$$

узлами

$$\boxed{s+1}\ \ldots\ \boxed{k-2}\ \boxed{k-1}\ \boxed{k}$$

при помощи циклической перестановки. В упр. 40 доказывается, что эти действия приведут к уменьшению цены (за исключением случая $l_{k-2} = l$). Однако поскольку согласно лемме Y цена не может уменьшаться, следовательно, $l_{k-2} = l$ и лемма доказана. ▌

Эти леммы показывают, что задача для $n+1$ весов $q_0, q_1, \ldots, q_n$ может быть сведена к задаче для n весов: сначала находим наименьший индекс k, такой, что $q_{k-1} \le q_{k+1}$, а затем — наибольшее $j < k$, для которого $q_{j-1} \ge q_{k-1} + q_k$. После этого мы удаляем q_{k-1} и q_k и вставляем сумму $q_{k-1} + q_k$ непосредственно после q_{j-1}. В специальном случае $j = 0$ или $k = n$, как вытекает из доказательства, мы должны действовать так, как если бы у нас были бесконечные веса q_{-1} и q_{n+1} слева и справа. Доказательства также показывают, что любое оптимальное дерево T', полученное на основе новых весов $(q'_0, \ldots, q'_{n-1})$, может быть переупорядочено в дерево T, в котором исходные веса $(q_0, \ldots, q_n)$ находятся в корректном порядке слева направо; более того, каждый вес появляется на одном и том же уровне как в дереве T, так и в дереве T'.

В качестве примера на рис. 18 показано построение дерева для весов q_k, представляющих относительные частоты появления букв ␣, A, B, ..., Z в английском тексте. Первые веса в этом списке равны

$$186,\ 64,\ 13,\ 22,\ 32,\ 103,\ \ldots,$$

и выполняются неравенства $186 > 13$, $64 > 22$, $13 \le 32$; следовательно, мы должны заменить $13, 22$ значением 35. В новой последовательности

$$186,\ 64,\ 35,\ 32,\ 103,\ \ldots$$

следует заменить $35, 32$ значением 67 и сдвинуть 67 левее 64, получив при этом последовательность

$$186,\ 67,\ 64,\ 103,\ \ldots.$$

Затем $67, 64$ превращаются в 131 и мы приступаем к исследованию весов, следующих за 103. После того как 27 исходных весов превращаются в один вес, $1\,000$, история комбинирования весов определяет бинарное дерево, которое и является решением поставленной задачи.

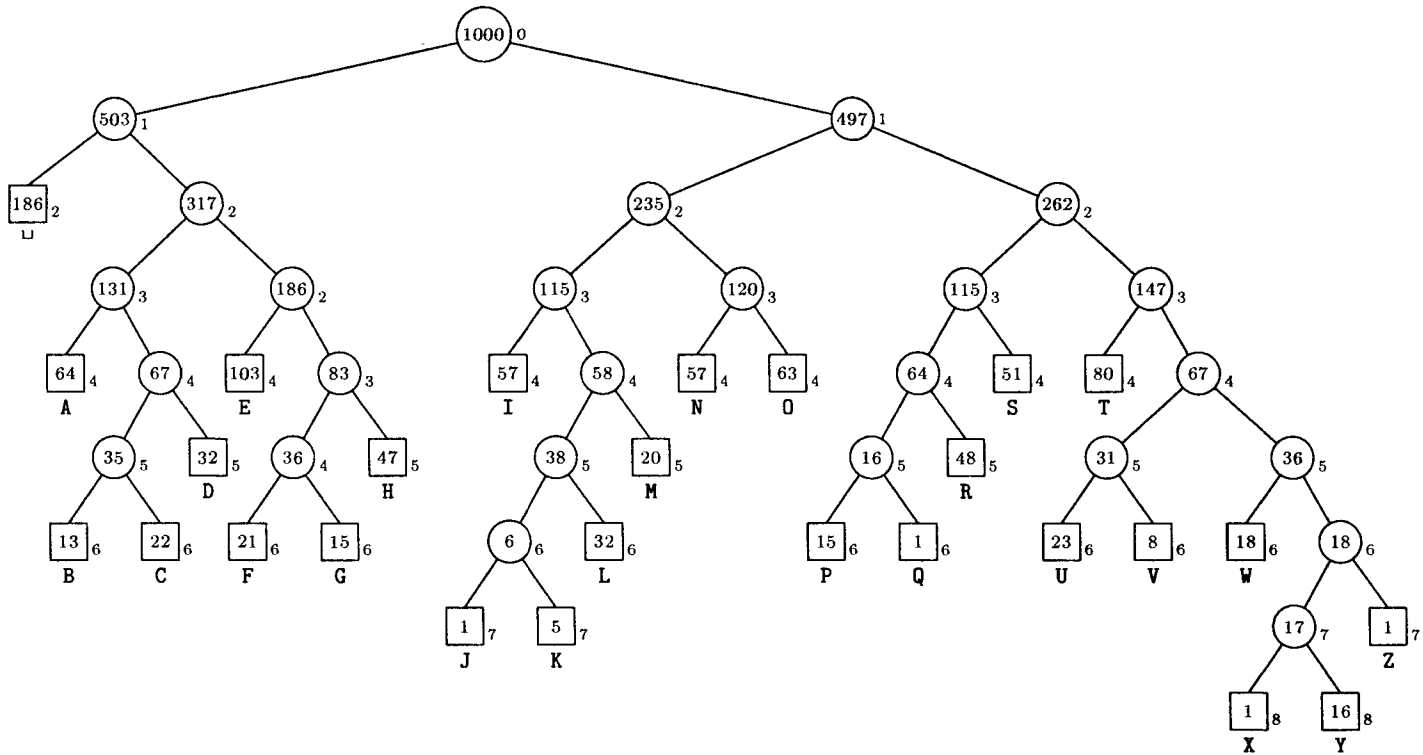


Рис. 19. Применение алгоритма Гарсия-Воча к данным о частотах появления букв; фаза 3.

Однако листья дерева на рис. 18 не находятся в правильном порядке, поскольку при каждом перемещении $q_{k-1} + q_k$ влево дерево тщательно запутывается (см. упр. 41). Тем не менее доказательство леммы Z гарантирует существование дерева, листья которого расположены в правильном порядке и на тех же уровнях, что и в "запутанном" дереве. Такое "распутанное" дерево показано на рис. 19; это оптимальное дерево получено по алгоритму Гарсия-Воча.

Алгоритм G (*Алгоритм Гарсия-Воча построения оптимальных бинарных деревьев*). Дана последовательность неотрицательных весов $w_0, w_1, \ldots, w_n$. Алгоритм позволяет построить бинарное дерево с n внутренними узлами, $\sum_{k=0}^{n} w_k l_k$ которого минимальна (здесь l_k представляет собой расстояние внешнего узла $\boxed{k}$ от корня дерева). Алгоритм использует массив из $2n+2$ узлов с адресами $\mathtt{X}_k$, где $0 \le k \le 2n+1$. Каждый узел имеет четыре поля, а именно — WT, LLINK, RLINK и LEVEL. Листья построенного дерева — это узлы $\mathtt{X}_0 \ldots \mathtt{X}_n$; внутренними узлами будут узлы $\mathtt{X}_{n+1} \ldots \mathtt{X}_{2n}$. Корневым узлом дерева является узел $\mathtt{X}_{2n}$, а узел $\mathtt{X}_{2n}$ используется в качестве временного. Алгоритм, кроме того, использует рабочий массив указателей $\mathtt{P}_0, \mathtt{P}_1, \ldots, \mathtt{P}_t$, где $t \le n+1$.

G1. [Начало фазы 1.] Установите $\mathtt{WT}(\mathtt{X}_k) \leftarrow w_k$ и $\mathtt{LLINK}(\mathtt{X}_k) \leftarrow \mathtt{RLINK}(\mathtt{X}_k) \leftarrow \Lambda$ для $0 \le k \le n$. Установите также $\mathtt{P}_0 \leftarrow \mathtt{X}_{2n+1}$, $\mathtt{WT}(\mathtt{P}_0) \leftarrow \infty$, $\mathtt{P}_1 \leftarrow \mathtt{X}_0$, $t \leftarrow 1$, $m \leftarrow n$. Затем выполните шаг G2 для $j = 1, 2, \ldots, n$ и перейдите к шагу G3.

G2. [Поглощение w_j.] (В этот момент у нас выполнены базовые условия

$$\mathtt{WT}(\mathtt{P}_{i-1}) > \mathtt{WT}(\mathtt{P}_{i+1}) \qquad \text{для } 1 \le i < t. \tag{31}$$

Другими словами, веса в рабочем массиве "попарно убывающие".) Выполните подпрограмму C, описанную ниже, несколько раз, пока не будет соблюдено условие $\mathtt{WT}(\mathtt{P}_{t-1}) > w_j$ (если это условие соблюдено изначально, то выполнять подпрограмму C не нужно). Затем установите $t \leftarrow t+1$ и $\mathtt{P}_t \leftarrow \mathtt{X}_j$.

G3. [Завершение фазы 1.] Выполните подпрограмму C несколько раз (возможно, ни разу), пока t не станет равным 1.

G4. [Фаза 2.] (Теперь $\mathtt{P}_1 = \mathtt{X}_{2n}$ — корень бинарного дерева и $\mathtt{WT}(\mathtt{P}_1) = w_0 + \cdots + w_n$.) Установите l_k равными расстояниям от узла $\mathtt{X}_k$ до узла $\mathtt{P}_1$ для $0 \le k \le n$ (см. упр. 43; на рис. 18 приведен пример построенного дерева; номера уровней показаны справа от узлов).

G5. [Фаза 3.] Изменяя связи $\mathtt{X}_{n+1}, \ldots, \mathtt{X}_{2n}$, постройте новое бинарное дерево с теми же уровнями l_k, но с листьями, расположенными в симметричном порядке $\mathtt{X}_0, \ldots, \mathtt{X}_n$ (см. упр. 44; пример полученного дерева показан на рис. 19). ▮

Подпрограмма C (*Объединение*). Эта подпрограмма является "сердцем" алгоритма Гарсия-Воча. Она объединяет два веса и смещает их на необходимое количество полей влево, обеспечивая выполнение условия "попарного убывания" (31).

C1. [Инициализация.] Установите $k \leftarrow t$.

C2. [Создание нового узла.] (В настоящий момент $k \ge 2$.) Установите $m \leftarrow m+1$, $\mathtt{LLINK}(\mathtt{X}_m) \leftarrow \mathtt{P}_{k-1}$, $\mathtt{RLINK}(\mathtt{X}_m) \leftarrow \mathtt{P}_k$, $\mathtt{WT}(\mathtt{X}_m) \leftarrow \mathtt{WT}(\mathtt{P}_{k-1}) + \mathtt{WT}(\mathtt{P}_k)$.

C3. [Сдвиг последующих узлов влево.] Установите $t \leftarrow t-1$, а затем $\mathtt{P}_{j+1} \leftarrow \mathtt{P}_j$ для $k \le j \le t$.

C4. [Сдвиг предыдущих узлов вправо.] Установите $j \leftarrow k-2$; затем, пока $\mathtt{WT}(\mathtt{P}_j) < \mathtt{WT}(\mathtt{X}_m)$, присваивайте $\mathtt{P}_{j+1} \leftarrow \mathtt{P}_j$ и $j \leftarrow j-1$.

C5. [Вставка нового узла.] Присвойте $\mathtt{P}_{j+1} \leftarrow \mathtt{X}_m$.

C6. [Все?] Если $j > 0$ и $\mathtt{WT}(\mathtt{P}_{j-1}) \le \mathtt{WT}(\mathtt{X}_m)$, присвойте $k \leftarrow j$ и вернитесь к шагу C2. ∎

Как указывалось выше, подпрограмме C может потребоваться $\Omega(n)$ шагов для создания и вставки нового узла, поскольку она использует последовательное хранение в памяти вместо связанных списков. Таким образом, общее время работы алгоритма G может составлять $\Omega(n^2)$. Однако более тщательная разработка применяемых структур данных может привести к использованию не более $O(n \log n)$ шагов (см. упр. 45). Фазы 2 и 3 требуют только $O(n)$ шагов.

Кляйтман (Kleitman) и Сакс (Saks) (*SIAM J. Algeb. Discr. Methods* **2** (1981), 142–146) доказали, что оптимальная взвешенная длина пути никогда не превышает значение оптимальной взвешенной длины пути, которая получается при перестановке q в "пилообразном" порядке:

$$q_0 \le q_2 \le q_4 \le \cdots \le q_{2\lfloor n/2 \rfloor} \le q_{2\lceil n/2 \rceil - 1} \le \cdots \le q_3 \le q_1 . \tag{32}$$

(Этот порядок представляет собой инверсию "органного" порядка, который обсуждался в упр. 6.1–18.) В последнем случае алгоритм Гарсия-Воча сводится к алгоритму Хаффмана на множестве весов $q_0 + q_1$, $q_2 + q_3$, ... , поскольку веса в рабочем массиве в действительности находятся в порядке невозрастания (а не только в "попарно убывающем" порядке, как в (31)). Следовательно, мы можем улучшить верхнюю границу в теореме M, не зная порядка весов.

Оптимальное бинарное дерево на рис. 19 имеет, помимо значения в теории поиска, большое прикладное значение в теории кодирования: используя 0 для левых ветвей дерева и 1 — для правых, мы получим следующие коды переменной длины.

␣	00	I	1000	R	11001	
A	0100	J	1001000	S	1101	
B	010100	K	1001001	T	1110	
C	010101	L	100101	U	111100	
D	01011	M	10011	V	111101	(33)
E	0110	N	1010	W	111110	
F	011100	O	1011	X	11111100	
G	011101	P	110000	Y	11111101	
H	01111	Q	110001	Z	1111111	

Таким образом, сообщение типа RIGHT ON может быть закодировано строкой

$$1100110000111010111111100010111010.$$

Декодирование слева направо выполняется просто и однозначно, несмотря на то что коды имеют различную длину — сама структура дерева показывает, когда заканчивается один символ и начинается другой. При данном методе кодирования сохраняется алфавитный порядок и используется в среднем около 4.2 бит для кодирования одного символа. Таким образом, этот код можно использовать для упаковки файлов данных без нарушения лексикографического порядка буквенной информации.

(Число 4.2 бит на символ минимально среди всех кодов, в которых используются бинарные деревья; оно может быть уменьшено до 4.1 бит на символ при отказе от алфавитного порядка. Уменьшение до 4.1 бит на символ с сохранением алфавитного порядка возможно при кодировании не отдельных символов, а пар символов.)

История и библиография. Рассмотренные в этом разделе методы поиска с использованием деревьев были открыты независимо несколькими исследователями в 50-е годы. В неопубликованном меморандуме, датированном августом 1952 года, А. И. Думи (A. I. Dumey) описал примитивный путь вставки в дерево.

Рассмотрим барабан с 2^n элементами, каждый из которых имеет бинарный адрес.

Следуйте описанной ниже программе.

1. Прочтите первый элемент и поместите его по адресу 2^{n-1}, т. е. в середину массива хранения.

2. Прочтите следующий элемент. Сравните его с первым.

3. Если он больше, поместите его по адресу $2^{n-1} + 2^{n-2}$. Если же он меньше, поместите его по адресу 2^{n-2}, и т. д.

Другая ранняя форма вставки в дерево была введена Д. Дж. Вилером (D. J. Wheeler), который допускал многопутевые разветвления (подобные тем, которые будут рассмотрены в разделе 6.2.4); еще один метод вставки в бинарное дерево был предложен К. М. Бернерсом-Ли (C. M. Berners-Lee) (см. *Comp. J.* **2** (1959), 5).

Первые опубликованные описания вставки в дерево принадлежат П. Ф. Виндли (P. F. Windley) (*Comp. J.* **3** (1960), 84–88), Э. Д. Буту (A. D. Booth) и Э. Дж. Т. Колину (A. J. T. Colin) (*Information and Control* **3** (1960), 327–334), а также Томасу Н. Хиббарду (Thomas N. Hibbard) (*JACM* **9** (1962), 13–28). По-видимому, каждый из авторов пришел к своему методу независимо от других, и в каждой статье среднее количество сравнений (6) выводится по-своему. Авторы также сосредоточивали свое внимание на разных аспектах алгоритма: Виндли подробно разбирал сортировку путем вставки в дерево, Бут и Колин исследовали влияние предварительного построения идеально сбалансированного дерева из первых $2^n - 1$ элементов (см. упр. 4), Хиббард предложил идею удаления и показал связь между анализом вставки в дерево и анализом быстрой сортировки.

Идеи *оптимальных* бинарных деревьев поиска первоначально были развиты для частного случая $p_1 = \cdots = p_n = 0$ в контексте бинарного кодирования алфавита, подобного (33). В очень интересной статье Э. Н. Гильберта (E. N. Gilbert) и Э. Ф. Мура (E. F. Moore) [*Bell System Tech. J.* **38** (1959), 933–968] обсуждается эта задача и ее связь с другими задачами кодирования. Гильберт и Мур доказали теорему М для специального случая $P = 0$ и заметили, что оптимальное дерево может быть построено за $O(n^3)$ шагов с помощью метода наподобие алгоритма К, но без использования соотношения монотонности (17). К. Ю. Айверсон (K. E. Iverson) [*A Programming Language* (Wiley, 1962), 142–144] независимо рассмотрел *другой* случай, когда все q_k равны нулю. Он предположил, что оптимальное дерево можно получить, выравнивая вероятности левого и правого поддеревьев; к сожалению, мы видели, что эта идея не срабатывает... Д. Э. Кнут (D. E. Knuth) [*Acta Informatica* **1** (1971), 14–25, 270] последовательно рассмотрел случаи произвольных весов p_k

и q_k и доказал, что алгоритм может быть сведен к $O(n^2)$ шагам; он также представил пример использования этих методов в компиляторе, где ключами дерева являлись "зарезервированные слова" ALGOL-подобного языка. Т. Ч. Ху (T. C. Hu) несколько лет изучал собственный алгоритм для случая $p_j = 0$; из-за сложности задачи было трудно найти строгое доказательство корректности алгоритма, однако в сотрудничестве с А. К. Таккером (A. C. Tucker) такое доказательство, в конце концов, было найдено [*SIAM J. Applied Math.* **21** (1971), 514–532]. Упрощения, приведшие к разработке алгоритма G, были найдены несколькими годами позже А. М. Гарсия (A. M. Garsia) и М. Л. Вочем (M. L. Wachs) [*SICOMP* **6** (1977), 622–642]; их доказательство, впрочем, было слишком сложным и запутанным. Леммы W, X, Y и Z, описанные выше, появились благодаря Дж. Х. Кингстону (J. H. Kingston) [*J. Algorithms* **9** (1988), 129–136]. (См. также статью Ху (Hu), Кляйтмана (Kleitman) и Тамаки (Tamaki) [*SIAM J. Applied Math.* **37** (1979), 246–256], в которой дано элементарное доказательство алгоритма Ху-Таккера и некоторые обобщения для других ценовых функций.)

Теорема B доказана Паулем Дж. Байером (Paul J. Bayer) [MIT/LCS/TM-69 (Mass. Inst. of Tech., 1975)], который также доказал теорему M (в ослабленной формулировке). Строгое доказательство указанной теоремы найдено К. Мельхорном (K. Mehlhorn) [*SICOMP* **6** (1977), 235–239].

УПРАЖНЕНИЯ

1. [*15*] Алгоритм T сформулирован только для непустых деревьев. Какие изменения должны быть внесены в алгоритм для корректной работы и в случае пустых деревьев?

2. [*20*] Измените алгоритм T таким образом, чтобы он работал с *правопрошитыми* деревьями (см. раздел 2.3.1; такие деревья упрощают выполнение симметричного обхода дерева).

▶ **3.** [*20*] В разделе 6.1 мы видели, что, внеся небольшие изменения в алгоритм последовательного поиска 6.1S, можно сделать его более быстрым (алгоритм 6.1Q). Можно ли воспользоваться похожим приемом для ускорения работы алгоритма T?

4. [*M24*] (Э. Д. Бут (A. D. Booth) и Э. Дж. Т. Колин (A. J. T. Colin).) Даны N ключей в случайном порядке. Предположим, что мы используем первые $2^n - 1$ из них для построения идеально сбалансированного дерева и размещаем 2^k ключей на уровне k для всех $0 \le k < n$; затем для вставки оставшихся ключей мы используем алгоритм T. Чему равно среднее число сравнений в случае успешного поиска? (*Указание.* Модифицируйте формулу (2).)

▶ **5.** [*M25*] Имеется $11! = 39\,916\,800$ различных способов упорядочения названий `CAPRICORN`, `AQUARIUS` и т. д. для их вставки в бинарное дерево поиска.

a) В результате скольких перестановок получится дерево, изображенное на рис. 10?

b) В результате скольких перестановок получится *вырожденное* дерево, в каждом узле `LLINK` или `RLINK` которого равны Λ?

6. [*M26*] Пусть P_{nk} — количество перестановок $a_1 a_2 \ldots a_n$ множества $\{1, 2, \ldots, n\}$, таких, что при использовании алгоритма T для вставки $a_1, a_2, \ldots, a_n$ в изначально пустое дерево для вставки элемента a_n потребуется ровно k сравнений. (В этой задаче мы пренебрегаем сравнениями, выполняемыми при вставке $a_1, \ldots, a_{n-1}$. В принятых в тексте обозначениях $C'_{n-1} = (\sum_k kP_{nk})/n!$, так как это среднее количество сравнений, которые выполняются в случае неудачного поиска в дереве, содержащем $n - 1$ элемент.)

a) Докажите, что $P_{(n+1)k} = 2P_{n(k-1)} + (n-1)P_{nk}$. (*Указание.* Посмотрите, не окажется ли в дереве a_{n+1} ниже, чем a_n.)

b) Найдите простую формулу для производящей функции $G_n(z) = \sum_k P_{nk} z^k$ и используйте ее для выражения P_{nk} через числа Стирлинга.

c) Чему равна *дисперсия* этого распределения?

7. [*M25*] (С. Р. Арора (S. R. Arora) и В. Т. Дент (W. T. Dent).) Чему равно среднее количество сравнений, необходимое алгоритму T для поиска m-го наибольшего элемента по его ключу после вставки n элементов в случайном порядке в первоначально пустое дерево?

8. [*M38*] Пусть $p(n, k)$ — вероятность того, что полная длина внутреннего пути дерева, построенного по алгоритму T из n случайным образом упорядоченных ключей, равна k (длина внутреннего пути дерева равна числу сравнений, производимых при сортировке путем вставки в дерево в процессе построения дерева).

a) Найдите рекуррентное соотношение, которое определяет соответствующую производящую функцию.

b) Вычислите дисперсию такого распределения. (При выполнении этого упражнения могут оказаться полезными некоторые упражнения из раздела 1.2.7.)

9. [*41*] Мы доказали, что поиск по дереву и вставка требуют порядка $2 \ln N$ сравнений при вставке ключей в случайном порядке. Однако на практике порядок вставки может быть не случайным. Проведите эмпирические исследования, чтобы выяснить, насколько хорошо вставка в дерево подходит для работы с реальными таблицами символов компилятора или ассемблера. Получится ли в результате вставки идентификаторов большой программы хорошо сбалансированное дерево поиска?

▶ **10.** [*22*] (Р. В. Флойд (R. W. Floyd).) Предположим, нас не интересует порядок ключей в дереве поиска; однако ожидается, что данные будут вводиться в неслучайном порядке. Обдумайте методы, которые позволят сохранить эффективность поиска, делая входные данные кажущимися случайными.

11. [*20*] Чему равно максимальное число присвоений `S ← LLINK(R)`, выполняемых на шаге D3 при удалении узла из дерева размером N?

12. [*M22*] Как часто в среднем происходит переход от шага D1 к шагу D4 при случайном удалении из случайного дерева, состоящего из N элементов? (См. доказательство теоремы H.)

▶ **13.** [*M23*] Если корень случайного дерева удаляется с помощью алгоритма D, останется ли получившееся в результате дерево случайным?

▶ **14.** [*22*] Докажите, что длина пути дерева, построенного согласно алгоритму D с дополнительным шагом D1$\frac{1}{2}$, никогда не превышает длину пути дерева, построенного без этого шага. В каком случае дополнительный шаг D1$\frac{1}{2}$ уменьшает длину пути?

15. [*23*] Пусть $a_1\, a_2\, a_3\, a_4$ — перестановка множества $\{1, 2, 3, 4\}$ и пусть $j = 1$, 2 или 3. Возьмем одноэлементное дерево с ключом a_1, выполним вставку элементов a_2, a_3 по алгоритму T, удалим a_j по алгоритму D и вставим a_4 согласно алгоритму T. Сколько деревьев из $4! \times 3$ возможных будут иметь вид I, II, III, IV и V соответственно (согласно классификации (13))?

▶ **16.** [*25*] Является ли операция удаления *коммутативной*? Иначе говоря, получится ли одно и то же дерево, если с помощью алгоритма D будет сначала удален узел X, а затем — Y и сначала удален узел Y, а затем — X?

17. [*25*] Покажите, что если в алгоритме D полностью заменить "левое" "правым", то его легко можно будет распространить на удаление данного узла из *правопрошитого* дерева с сохранением необходимых нитей (см. упр. 2).

18. [*M21*] Покажите, что, используя закон Зипфа, можно получить (12).

19. [*M23*] Чему равно приближенное среднее количество сравнений (11), если вероятности входных данных подчиняются закону "80–20", определенному в 6.1–(11)?

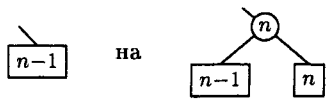
20. [*M20*] Предположим, что мы вставляем ключи в дерево в порядке уменьшения их частот $p_1 \geq p_2 \geq \cdots \geq p_n$. Не может ли такое дерево оказаться существенно хуже оптимального?

21. [*M20*] Если p, q, r — случайно выбранные вероятности, такие, что $p+q+r=1$, чему равны вероятности того, что деревья I, II, III, IV, V из (13) оптимальны? (Рассмотрите соотношения площадей соответствующих областей на рис. 14.)

22. [*M20*] Докажите, что при выполнении шага K4 алгоритма K $r[i,j-1]$ никогда не превышает $r[i+1,j]$.

▶ **23.** [*M23*] Найдите оптимальное бинарное дерево поиска для случая $N = 40$ с весами $p_1 = 9$, $p_2 = p_3 = \cdots = p_{40} = 1$, $q_0 = q_1 = \cdots = q_{40} = 0$ (не используя компьютер).

24. [*M25*] Пусть $p_n = q_n = 0$, а все остальные веса неотрицательны. Докажите, что оптимальное дерево для $(p_1, \ldots, p_n;\ q_0, \ldots, q_n)$ может быть получено посредством замены



в любом оптимальном для $(p_1, \ldots, p_{n-1};\ q_0, \ldots, q_{n-1})$ дереве.

25. [*M20*] Пусть A и B — непустые множества действительных чисел. Определим, что $A \leq B$, если выполняется следующее:

$$\text{из} \qquad (a \in A,\ b \in B \text{ и } b < a) \qquad \text{следует} \qquad (a \in B \text{ и } b \in A).$$

a) Докажите, что это соотношение транзитивно на непустых множествах.
b) Докажите или опровергните, что $A \leq B$ тогда и только тогда, когда $A \leq A \cup B \leq B$.

26. [*M22*] Пусть $(p_1, \ldots, p_n;\ q_0, \ldots, q_n)$ — неотрицательные веса и $p_n + q_n = x$. Докажите, что при x, изменяющемся от 0 до ∞, и при постоянных $(p_1, \ldots, p_{n-1};\ q_0, \ldots, q_{n-1})$ цена $c(0,n)$ оптимального бинарного дерева поиска является выпуклой, непрерывной, кусочно-линейной функцией x с целыми угловыми коэффициентами. Другими словами, докажите, что существуют положительные целые числа $l_0 > l_1 > \cdots > l_m$ и действительные константы $0 = x_0 < x_1 < \cdots < x_m < x_{m+1} = \infty$ и $y_0 < y_1 \cdots < y_m$, такие, что $c(0,n) = y_h + l_h x$ при $x_h \leq x \leq x_{h+1}$, $0 \leq h \leq m$.

27. [*M33*] Цель данного упражнения состоит в доказательстве того, что множество корней $R(i,j)$ оптимальных бинарных деревьев удовлетворяет соотношению

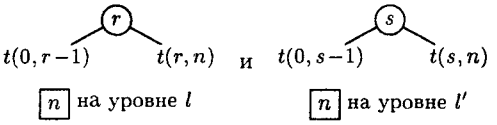
$$R(i,j-1) \leq R(i,j) \leq R(i+1,j) \qquad \text{для } j-i \geq 2,$$

где отношение $\leq$ введено в упр. 25 при неотрицательных весах $(p_1, \ldots, p_n;\ q_0, \ldots, q_n)$. Доказательство проводится по индукции по $j-i$; наша задача состоит в том, чтобы доказать, что $R(0,n-1) \leq R(0,n)$ при $n \geq 2$ в предположении справедливости соотношения при $j-i<n$. (В силу симметрии отсюда следует, что $R(0,n) \leq R(1,n)$.)

a) Докажите, что $R(0,n-1) \leq R(0,n)$ при $p_n = q_n = 0$ (см. упр. 24).
b) Пусть $p_n + q_n = x$. Воспользуемся обозначениями упр. 26. Пусть R_h — множество оптимальных корней $R(0,n)$ при $x_h < x < x_{h+1}$, а R'_h — множество оптимальных корней для $x = x_h$. Докажите, что

$$R'_0 \leq R_0 \leq R'_1 \leq R_1 \leq \cdots \leq R'_m \leq R_m.$$

Отсюда из п. (а) и упр. 25 следует, что $R(0,n-1) \le R(0,n)$ для всех x. (*Указание.* Рассмотрите случай $x = x_h$ и предположите, что деревья



оптимальны с $s < r$ и $l \ge l'$. Воспользуйтесь предположением индукции для доказательства того, что существует оптимальное дерево с корнем r, у которого $\boxed{n}$ располагается на уровне l', а также оптимальное дерево с корнем s и узлом $\boxed{n}$ на уровне l.)

28. [*24*] Используйте некоторый макроязык для определения макроса "оптимальный бинарный поиск", параметром которого является вложенная спецификация оптимального бинарного дерева.

29. [*40*] Каково *наихудшее* бинарное дерево поиска для 31 наиболее употребительного английского слова (эти слова представлены вместе с частотами их появления на рис. 12)?

30. [*M34*] Докажите, что цена оптимальных бинарных деревьев поиска удовлетворяет "квадрантному неравенству" $c(i,j) - c(i,j-1) \ge c(i+1,j) - c(i+1,j-1)$ при $j \ge i+2$.

31. [*M35*] (К. Ч. Тан (K. C. Tan).) Докажите, что среди всех распределений вероятностей $(p_1,\ldots,p_n;\ q_0,\ldots,q_n)$ $(p_1+\cdots+p_n+q_0+\cdots+q_n = 1)$ дерево с минимальной ценой наиболее дорого при $p_i = 0$ для всех i, $q_j = 0$ для всех четных j и $q_j = 1/\lceil n/2\rceil$ для всех нечетных j.

▶ **32.** [*M25*] Пусть $n+1 = 2^m+k$, где $0 \le k \le 2^m$. Существует ровно $\binom{2^m}{k}$ бинарных деревьев, в которых все внешние узлы расположены на уровнях m и $m+1$. Покажите, что среди всех этих деревьев мы получим одно с минимальной ценой для весов $(p_1,\ldots,p_n;q_0,\ldots,q_n)$, если примéним алгоритм К к весам $(p_1,\ldots,p_n;M+q_0,\ldots,M+q_n)$ при достаточно большом M.

33. [*M41*] Для нахождения бинарного дерева поиска, минимизирующего время работы программы Т, следует минимизировать величину $7C + C1$, а не просто количество сравнений C. Разработайте алгоритм для поиска оптимальных бинарных деревьев поиска, когда левые и правые ветви дерева имеют разные цены. (В случае, когда "правая" цена в два раза больше "левой", а частоты всех узлов одинаковы, оптимальными становятся деревья Фибоначчи [см. L. E. Stanfel, *JACM* **17** (1970), 508–517].) На машинах, которые не могут получать результат сравнения из трех возможных (больше, меньше, равно), программа, реализующая алгоритм Т, вынуждена производить на шаге Т2 два сравнения — меньше и больше. Б. Шейл (B. Sheil) и В. Р. Пратт (V. R. Pratt) обнаружили, что эти сравнения не требуют одного и того же ключа и может оказаться предпочтительным бинарное дерево, внутренние узлы которого определяют либо проверку равенства, либо проверку "меньше, чем", но не обе. Такая ситуация может оказаться интересной альтернативой поставленной задаче.

34. [*HM21*] Покажите, что асимптотическое значение полиномиального коэффициента

$$\binom{N}{p_1N,\ p_2N,\ \ldots,\ p_nN}$$

при $N \to \infty$ стремится к энтропии $H(p_1,p_2,\ldots,p_n)$.

35. [*HM22*] Завершите доказательство теоремы В, доказав справедливость неравенства (24).

▶ **36.** [*HM25*] (Клод Шеннон (Claude Shannon).) Пусть X и Y — случайные величины, принимающие конечные множества значений $\{x_1,\ldots,x_m\}$ и $\{y_1,\ldots,y_n\}$. Введем обозначения $p_i = \Pr(X = x_i)$, $q_j = \Pr(Y = y_j)$, $r_{ij} = \Pr(X = x_i \text{ и } Y = y_j)$; кроме того, положим, что $H(X) = H(p_1,\ldots,p_m)$ и $H(Y) = H(q_1,\ldots,q_n)$ представляют собой энтропии X и Y

по отдельности, а $H(XY) = H(r_{11}, \ldots, r_{mn})$ — энтропия их совместного распределения. Докажите, что

$$H(X) \le H(XY) \le H(X) + H(Y).$$

(*Указание.* Если f — вогнутая функция, то $\mathrm{E}\, f(X) \le f(\mathrm{E}\, X)$.)

37. [*HM26*] (П. Дж. Байер (P. J. Bayer), 1975.) Предположим, что $(P_1, \ldots, P_n)$ представляет собой случайное распределение вероятностей, а именно — случайную точку в $(n-1)$-мерном симплексе, определенную величинами $P_k \ge 0$, $1 \le k \le n$, где $P_1 + \cdots + P_n = 1$ (или, что то же самое, $(P_1, \ldots, P_n)$ представляют собой множество случайных *промежутков*, о которых говорилось в упр. 3.3.2–26). Чему равно ожидаемое значение энтропии $H(P_1, \ldots, P_n)$?

38. [*M20*] Поясните, почему теорема M справедлива в общем случае, хотя мы доказали ее только для случая $s_0 < s_1 < s_2 < \cdots < s_n$.

▶ **39.** [*M25*] Пусть $w_1, \ldots, w_n$ представляют собой неотрицательные веса ($w_1 + \cdots + w_n = 1$). Докажите, что взвешенная длина пути дерева Хаффмана, построенного в разделе 2.3.4.5, меньше, чем $H(w_1, \ldots, w_n) + 1$. (*Указание.* Обратитесь к доказательству теоремы M.)

40. [*M26*] Завершите доказательство леммы Z.

41. [*21*] На рис. 18 показано строение весьма запутанного бинарного дерева. Перечислите его листья в порядке слева направо.

42. [*23*] Объясните, почему в подпрограмме C сохраняется справедливость условия (31).

43. [*20*] Поясните, как эффективно реализовать фазу 2 алгоритма Гарсия-Воча.

▶ **44.** [*25*] Поясните, как эффективно реализовать фазу 3 алгоритма Гарсия-Воча: постройте бинарное дерево с уровнями l_0, l_1, ..., l_n листьев в симметричном порядке.

▶ **45.** [*30*] Поясните, как реализовать подпрограмму C так, чтобы общее время работы алгоритма Гарсия-Воча не превышало $O(n \log n)$.

46. [*M30*] (Ч. К. Вонг (C. K. Wong) и Ши-Куо Чанг (Shi-Kuo Chang).) Рассмотрим схему, в которой бинарное дерево поиска строится согласно алгоритму T, но, когда количество узлов достигает значений вида $2^n - 1$, дерево реорганизуется в идеально сбалансированное дерево с 2^k узлами на уровне k, $0 \le k < n$. Докажите, что общее количество сравнений, выполненных при построении такого дерева, в среднем равно $N \lg N + O(N)$. (Нетрудно показать, что время, необходимое для реорганизации дерева, составляет $O(N)$.)

47. [*M40*] Обобщите теоремы B и M для t-арных деревьев. По возможности рассмотрите случай, когда цены ветвей неодинаковы, как в упр. 33.

48. [*M47*] Проведите точный анализ устойчивого состояния бинарного дерева поиска при случайных вставках и удалениях.

49. [*HM42*] Проанализируйте среднюю высоту бинарного дерева поиска.

6.2.3. Сбалансированные деревья

Алгоритм вставки в дерево, который мы только что изучили, дает хорошие результаты при использовании случайных входных данных, но все же существует неприятная возможность того, что при этом будет построено вырожденное дерево. Можно было бы разработать алгоритм, поддерживающий дерево в оптимальном состоянии все время, однако это, к сожалению, очень сложная задача. Другая идея заключается в том, чтобы хранить общую длину пути и полностью реорганизовывать дерево, когда она превышает, например, $5N \lg N$. Тем не менее при таком подходе потребовалось бы порядка $\sqrt{N/2}$ реорганизаций дерева в процессе его построения.

Очень красивое решение проблемы поддержания дерева поиска в хорошем состоянии было открыто в 1962 году двумя советскими математиками — Г. М. Адельсон-Вельским и Е. М. Ландисом [*ДАН СССР* **146** (1962), 263–266; английский перевод *Soviet Math.* **3**, 1259–1263]. Их метод требует всего лишь двух дополнительных битов на узел и никогда не использует более $O(\log N)$ операций для поиска по дереву или вставки элемента. Предложенный подход также дает общую технологию представления *линейных списков* длиной N таким образом, что любая из следующих операций может быть выполнена за время $O(\log N)$:

i) поиск элемента по заданному ключу;

ii) поиск k-го элемента по заданному k;

iii) вставка элемента в определенное место;

iv) удаление определенного элемента.

При последовательном расположении элементов линейных списков операции (i) и (ii) выполняются очень эффективно, в то время как операции (iii) и (iv) требуют порядка N шагов. С другой стороны, при использовании связанных элементов эффективно будут выполняться операции (iii) и (iv), а операции (i) и (ii) потребуют порядка N шагов. Представление линейных списков в виде дерева позволяет выполнить *все четыре* операции за $O(\log N)$ шагов. При этом можно более или менее эффективно выполнять и другие операции, например сцепление списка из M элементов со списком из N элементов за $O\big(\log(M+N)\big)$ шагов.

Метод, предоставляющий все эти преимущества, будем называть *сбалансированными деревьями* (многие авторы используют термин *AVL-деревья* — по первым буквам фамилий авторов). Предыдущий абзац служит рекламой сбалансированных деревьев — эдакой панацеи от всех бед. Имея в руках такой мощный метод, можно смело выбросить на помойку все прочие методы! Однако не торопитесь — сначала сбалансируйте свое отношение к сбалансированным деревьям. Ведь, например, если все четыре операции не нужны, нас может удовлетворить менее универсальный, но более быстрый (для данного конкретного случая) и проще программируемый метод. Кроме того, сбалансированные деревья хороши при наличии большого количества элементов. Судите сами: если есть эффективный метод, который требует $64 \lg N$ единиц времени, и неэффективный, которому необходимо $2N$ единиц, то при $N \le 256$ следует использовать неэффективный метод, который при этом становится более эффективным... С другой стороны, при очень больших N хранение данных во *внутренней* памяти становится невозможным, а при использовании внешних файлов с прямым доступом более эффективны другие методы, о которых речь пойдет в разделе 6.2.4. Впрочем, оперативная память становится все дешевле, а значит, сбалансированные деревья становятся все более и более важным орудием в руках программиста.

Высота дерева определяется как его максимальный уровень, длина самого длинного пути от корня к внешнему узлу. Бинарное дерево называется *сбалансированным*, если высота левого поддерева любого узла отличается не более чем на ± 1 от высоты правого поддерева. На рис. 20 показано сбалансированное дерево высотой 5 с 17 внутренними узлами; *фактор сбалансированности* обозначен внутри каждого узла знаками "+", "•" и "−" в соответствии с величиной разности между высотами правого и левого поддеревьев (+1, 0 или −1). Дерево Фибоначчи на рис. 8

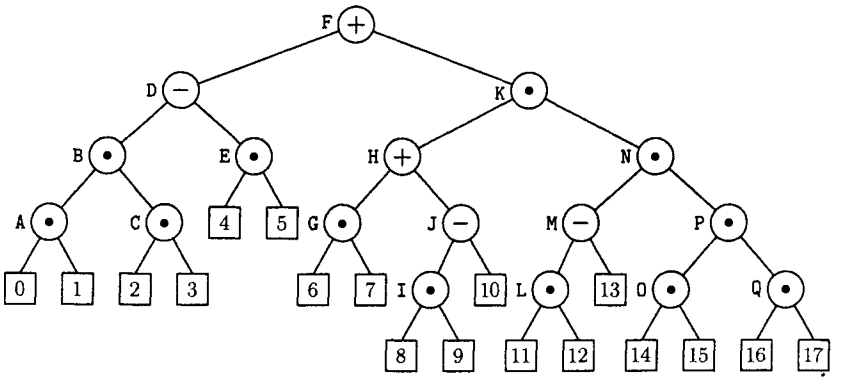


Рис. 20. Сбалансированное бинарное дерево.

в разделе 6.2.1 служит еще одним примером сбалансированного бинарного дерева высотой 5 с 12 внутренними узлами; большинство факторов сбалансированности в этом дереве равны -1. Дерево знаков зодиака на рис. 10 в разделе 6.2.2 *не* сбалансировано, поскольку для узлов `AQUARIUS` и `GEMINI` нарушено условие для величины разности высот поддеревьев.

Такое определение сбалансированности представляет собой компромисс между *оптимальными* бинарными деревьями (все внешние узлы которых должны размещаться на двух соседних уровнях) и *произвольными* бинарными деревьями. Отсюда естественным образом возникает вопрос "Насколько далеко от оптимального может оказаться сбалансированное дерево?". Ответ на него гласит, что путь поиска по сбалансированному дереву не превышает пути поиска по оптимальному дереву более чем на 45%.

Теорема А (Адельсон-Вельский и Ландис). *Высота сбалансированного дерева с N внутренними узлами ограничена значениями* $\lg(N+1)$ *и* $1.4405\lg(N+2) - 0.3277$.

Доказательство. Бинарное дерево высотой h, очевидно, не может иметь более 2^h внешних узлов; таким образом, $N+1 \le 2^h$, т. е. $h \ge \lceil \lg(N+1) \rceil$ для любого бинарного дерева.

Для поиска максимального значения h рассмотрим проблему с другой стороны и зададимся вопросом о минимальном количестве узлов в сбалансированном дереве высотой h. Пусть T_h — такое дерево с наименьшим возможным количеством узлов, тогда одно из поддеревьев корня, например левое, имеет высоту $h-1$, а правое — $h-1$ или $h-2$. Поскольку дерево T_h должно иметь минимальное количество узлов, можно считать, что в силу определения T_h левое поддерево представляет собой дерево T_{h-1}, а правое — T_{h-2}. Таким образом, наименьшее возможное количество узлов среди всех сбалансированных деревьев имеет *дерево Фибоначчи* (см. определение деревьев Фибоначчи в разделе 6.2.1). Следовательно,

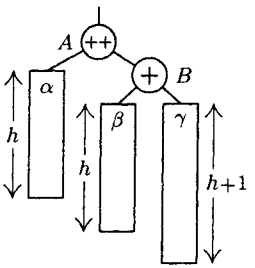
$$N \ge F_{h+2} - 1 > \phi^{h+2}/\sqrt{5} - 2$$

и требуемый результат получается так же, как и следствие из теоремы 4.5.3F. ∎

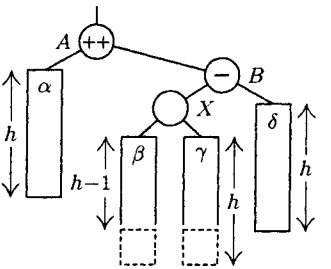
Как можно видеть из доказательства теоремы, для поиска в сбалансированном дереве будет требоваться больше 25 сравнений, только если дерево содержит хотя бы $F_{28} - 1 = 317\,810$ узлов.

Рассмотрим, что происходит при вставке нового узла в сбалансированное дерево с использованием алгоритма вставки в дерево 6.2.2T. Дерево, показанное на рис. 20, остается сбалансированным, если новый узел занимает место $\boxed{4}$, $\boxed{5}$, $\boxed{6}$, $\boxed{7}$, $\boxed{10}$ или $\boxed{13}$, но в других случаях потребуется определенная корректировка. Проблемы начинаются, когда имеется узел с фактором сбалансированности $+1$, правое поддерево которого после вставки становится выше (или, если фактор сбалансированности равен -1, выше становится левое поддерево). Нетрудно видеть, что неприятности возникают только в двух случаях.

Случай 1



Случай 2

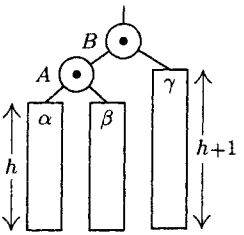


(1)

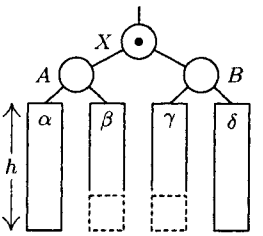
(Два других неприятных случая могут быть получены из указанных при зеркальном отражении относительно вертикальной оси.) На этой диаграмме большие прямоугольники α, β, γ, δ представляют поддеревья с соответствующими высотами. Случай 1 имеет место, когда новый элемент увеличивает высоту правого поддерева узла B с h до $h+1$, а случай 2 — когда увеличивается высота левого поддерева узла B. В последнем случае либо $h = 0$ (когда X представляет собой новый узел), либо узел X имеет два поддерева с высотами $(h-1, h)$ или $(h, h-1)$.

Выполнив простые преобразования, можно восстановить баланс в обоих случаях; при этом симметричный порядок узлов дерева сохранится.

Случай 1



Случай 2



(2)

В случае 1 мы просто "поворачиваем" дерево налево, присоединяя β к A вместо B. Такое преобразование подобно применению ассоциативного закона к алгебраической формуле, заменяющему $\alpha(\beta\gamma)$ на $(\alpha\beta)\gamma$. В случае 2 используется двойной поворот: сначала (X, B) поворачивается направо, затем (A, X) — налево. В обоих случаях в дереве требуется изменить лишь несколько ссылок. Кроме того, новые деревья имеют высоту $h+2$, точно равную высоте, которая была до вставки. Следовательно, остаток дерева (если таковой имеется), который изначально находился над узлом A, всегда остается сбалансированным.

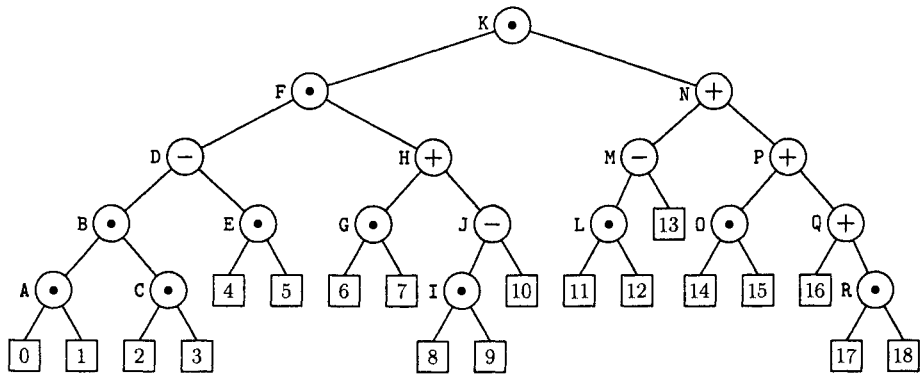


Рис. 21. Дерево, показанное на рис. 20, после вставки нового ключа R и выполнения балансировки.

Например, если вставить новый узел в позицию $\boxed{17}$ в дерево, представленное на рис. 20, после однократного поворота получится сбалансированное дерево, показанное на рис. 21 (случай 1). Обратите внимание, что при этом изменились некоторые факторы сбалансированности.

Детали этой процедуры вставки могут быть разработаны различными способами. На первый взгляд кажется, что избежать использования вспомогательного стека невозможно, поскольку требуется запоминать узлы, затронутые при вставке. Однако описанный ниже алгоритм позволяет обойтись без стека и выиграть при этом в скорости работы, если учесть, что фактор сбалансированности узла B в (1) был до вставки равен нулю.

Алгоритм А (*Поиск со вставкой по сбалансированному дереву*). Дана таблица записей в форме сбалансированного бинарного дерева, описанного выше. Алгоритм (рис. 22) производит поиск данного аргумента K. Если K отсутствует в таблице, в соответствующее место в дереве вставляется узел, в котором содержится K, и дерево при необходимости ребалансируется.

Предполагается, что, как и в алгоритме 6.2.2T, узлы дерева имеют поля KEY, LLINK и RLINK. Кроме того, имеется новое поле

B(P) = фактор сбалансированности узла NODE(P),

разность высот правого и левого поддеревьев. В нем всегда содержится только одно из трех значений: $+1$, 0 или -1. На вершине дерева по адресу HEAD расположен специальный узел, значение RLINK(HEAD) которого указывает на корень дерева, а LLINK(HEAD) используется для хранения полной высоты дерева (знание высоты не является необходимым для этого алгоритма, однако полезно при конкатенации, которая будет обсуждаться ниже). Также полагается, что дерево *непустое*, т. е. RLINK(HEAD) $\neq \Lambda$.

Для удобства описания в алгоритме используется обозначение LINK(a,P) как синоним для LLINK(P) при $a = -1$ и для RLINK(P) при $a = +1$.

A1. [Инициализация.] Установить T $\leftarrow$ HEAD, S $\leftarrow$ P $\leftarrow$ RLINK(HEAD). (Переменная-указатель P будет двигаться вниз по дереву; S будет указывать место, где

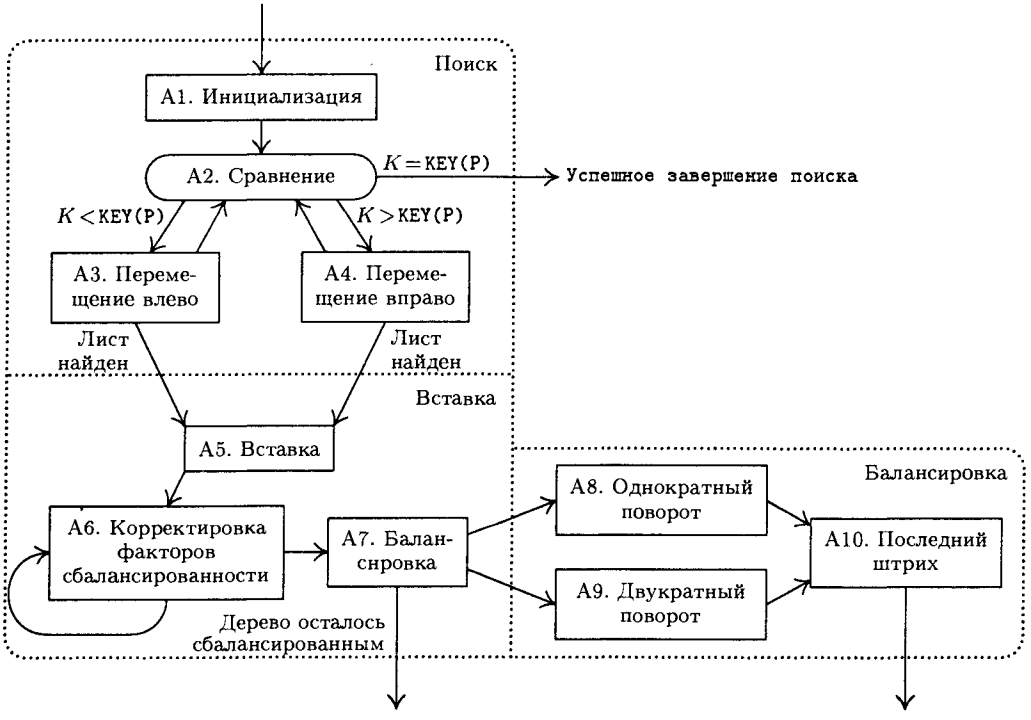


Рис. 22. Поиск со вставкой по сбалансированному дереву.

может потребоваться ребалансировка, а T всегда указывает на родительский по отношению к S узел.)

A2. [Сравнение.] Если $K <$ KEY(P), перейти к шагу A3; если $K >$ KEY(P), перейти к шагу A4; если $K =$ KEY(P), поиск успешно завершен.

A3. [Перемещение влево.] Установить Q $\leftarrow$ LLINK(P). Если Q $= \Lambda$, присвоить Q $\Leftarrow$ AVAIL и LLINK(P) $\leftarrow$ Q и перейти к шагу A5. В противном случае, если B(Q) $\neq 0$, присвоить T $\leftarrow$ P и S $\leftarrow$ Q. И, наконец, присвоить P $\leftarrow$ Q и вернуться к шагу A2.

A4. [Перемещение вправо.] Установить Q $\leftarrow$ RLINK(P). Если Q $= \Lambda$, присвоить Q $\Leftarrow$ AVAIL и RLINK(P) $\leftarrow$ Q и перейти к шагу A5. В противном случае, если B(Q) $\neq 0$, присвоить T $\leftarrow$ P и S $\leftarrow$ Q. И, наконец, присвоить P $\leftarrow$ Q и вернуться к шагу A2. (Последняя часть этого шага может быть объединена с последней частью шага A3.)

A5. [Вставка.] (Мы только что связали новый узел NODE(Q) с деревом; теперь следует инициализировать его поля.) Установить KEY(Q) $\leftarrow K$, LLINK(Q) $\leftarrow$ RLINK(Q) $\leftarrow \Lambda$ и B(Q) $\leftarrow 0$.

A6. [Корректировка факторов сбалансированности.] (В настоящий момент следует изменить факторы сбалансированности узлов между S и Q с 0 на ± 1.) Если $K <$ KEY(S), установить $a \leftarrow -1$; в противном случае установить $a \leftarrow +1$. Затем присвоить R $\leftarrow$ P $\leftarrow$ LINK(a,S) и при необходимости повторить следую-

щую операцию несколько раз, пока P не станет равным Q: если $K <$ KEY(P), установить B(P) $\leftarrow -1$ и P $\leftarrow$ LLINK(P); если $K >$ KEY(P), установить B(P) $\leftarrow +1$ и P $\leftarrow$ RLINK(P). (Если $K =$ KEY(P), то P $=$ Q и перейти к следующему шагу.)

A7. [Балансировка.] На этом шаге возможны несколько вариантов.

i) Если B(S) $= 0$ (дерево стало выше), установить B(S) $\leftarrow a$, LLINK(HEAD) $\leftarrow$ LLINK(HEAD) $+ 1$ и прекратить выполнение алгоритма.

ii) Если B(S) $= -a$ (дерево стало более сбалансированным), установить B(S) $\leftarrow 0$ и прекратить выполнение алгоритма.

iii) Если B(S) $= a$ (дерево разбалансировано), перейти к шагу A8 при B(R) $= a$ или к шагу A9 при B(R) $= -a$.
(Случай (iii) соответствует ситуации, приведенной в (1), при $a = +1$; S и R указывают на узлы A и B соответственно, LINK($-a$,S) указывает на α и т. д.)

A8. [Однократный поворот.] Установить P $\leftarrow$ R, LINK(a,S) $\leftarrow$ LINK($-a$,R), LINK($-a$,R) $\leftarrow$ S, B(S) $\leftarrow$ B(R) $\leftarrow 0$. Перейти к шагу A10.

A9. [Двукратный поворот.] Установить P $\leftarrow$ LINK($-a$,R), LINK($-a$,R) $\leftarrow$ LINK(a,P), LINK(a,P) $\leftarrow$ R, LINK(a,S) $\leftarrow$ LINK($-a$,P), LINK($-a$,P) $\leftarrow$ S; присвоить сначала

$$(\mathtt{B(S)},\mathtt{B(R)}) \leftarrow \begin{cases} (-a,0), & \text{если } \mathtt{B(P)} = a; \\ (0,0), & \text{если } \mathtt{B(P)} = 0; \\ (0,a), & \text{если } \mathtt{B(P)} = -a; \end{cases} \tag{3}$$

а затем — B(P) $\leftarrow 0$.

A10. [Последний штрих.] (Балансирующее преобразование (1) в (2) завершено; P указывает на корень нового поддерева, а T — на родительский по отношению к корню старого поддерева узел S.) Если S $=$ RLINK(T), следует установить RLINK(T) $\leftarrow$ P; в противном случае следует установить LLINK(T) $\leftarrow$ P. ▌

Этот алгоритм достаточно длинный, однако разделяется на три простые части: на шагах A1–A4 осуществляется поиск, на шагах A5–A7 — вставка нового узла и на шагах A8–A10 при необходимости ребалансируется дерево. Тот же алгоритм может использоваться и для *прошитых* деревьев (см. упр. 6.2.2–2).

Известно, что для этого алгоритма требуется около $C \log N$ единиц времени, где C — некоторая константа. Однако следует оценить ее величину, чтобы знать, при каких N использование сбалансированных деревьев становится эффективнее других алгоритмов. Приведенная ниже MIX-реализация алгоритма позволяет приступить к решению этого вопроса.

Программа A (*Поиск со вставкой по сбалансированному дереву*). Эта программа, реализующая алгоритм A, использует узлы дерева в следующем формате:

B	LLINK	RLINK
KEY		

; (4)

rA $\equiv K$, rI1 $\equiv$ P, rI2 $\equiv$ Q, rI3 $\equiv$ R, rI4 $\equiv$ S, rI5 $\equiv$ T. Код шагов A7–A9 дублируется, так что значение a используется в программе в неявном виде.

```
01 B     EQU  0:1
02 LLINK EQU  2:3
```

03	RLINK	EQU	4:5		
04	START	LDA	K	1	*A1. Инициализация.*
05		ENT5	HEAD	1	T ← HEAD.
06		LD2	0,5(RLINK)	1	Q ← RLINK(HEAD).
07		JMP	2F	1	Переход к A2 с S ← P ← Q.
08	4H	LD2	0,1(RLINK)	$C2$	*A4. Перемещение вправо.* Q ← RLINK(P).
09		J2Z	5F	$C2$	Переход к шагу A5 при Q = Λ.
10	1H	LDX	0,2(B)	$C-1$	rX ← B(Q).
11		JXZ	*+3	$C-1$	Переход, если B(Q) = 0.
12		ENT5	0,1	$D-1$	T ← P.
13	2H	ENT4	0,2	D	S ← Q.
14		ENT1	0,2	C	P ← Q.
15		CMPA	1,1	C	*A2. Сравнение.*
16		JG	4B	C	Переход к шагу A4, если $K >$ KEY(P).
17		JE	SUCCESS	$C1$	Выход при $K =$ KEY(P).
18		LD2	0,1(LLINK)	$C1-S$	*A3. Перемещение влево.* Q ← LLINK(P).
19		J2NZ	1B	$C1-S$	Переход при Q ≠ Λ.
20	5H	LD2	AVAIL	$1-S$	*A5. Вставка.*
21		J2Z	OVERFLOW	$1-S$	
22		LDX	0,2(RLINK)	$1-S$	
23		STX	AVAIL	$1-S$	Q ⇐ AVAIL.
24		STA	1,2	$1-S$	KEY(Q) ← K.
25		STZ	0,2	$1-S$	LLINK(Q) ← RLINK(Q) ← Λ.
26		JL	1F	$1-S$	$K <$ KEY(P)?
27		ST2	0,1(RLINK)	A	RLINK(P) ← Q
28		JMP	*+2	A	
29	1H	ST2	0,1(LLINK)	$1-S-A$	LLINK(P) ← Q.
30	6H	CMPA	1,4	$1-S$	*A6. Корректировка фактора сбалансированности.*
31		JL	*+3	$1-S$	Переход при $K <$ KEY(S).
32		LD3	0,4(RLINK)	E	R ← RLINK(S).
33		JMP	*+2	E	
34		LD3	0,4(LLINK)	$1-S-E$	R ← LLINK(S).
35		ENT1	0,3	$1-S$	P ← R.
36		ENTX	-1	$1-S$	rX ← −1.
37		JMP	1F	$1-S$	Переход к циклу сравнения.
38	4H	JE	7F	$F2+1-S$	Переход к шагу A7, если $K =$ KEY(P).
39		STX	0,1(1:1)	$F2$	B(P) ← +1 (было +0).
40		LD1	0,1(RLINK)	$F2$	P ← RLINK(P).
41	1H	CMPA	1,1	$F+1-S$	
42		JGE	4B	$F+1-S$	Переход при $K \geq$ KEY(P).
43		STX	0,1(B)	$F1$	B(P) ← −1.
44		LD1	0,1(LLINK)	$F1$	P ← LLINK(P).
45		JMP	1B	$F1$	Переход к циклу сравнения.
46	7H	LD2	0,4(B)	$1-S$	*A7. Балансировка.* rI2 ← B(S).
47		STZ	0,4(B)	$1-S$	B(S) ← 0.
48		CMPA	1,4	$1-S$	
49		JG	A7R	$1-S$	Переход к подпрограмме $a = +1$ при $K >$ KEY(S).
50	A7L	J2P	DONE	$U1$	Выход при rI2 $= -a$.
51		J2Z	7F	$G1+J1$	Переход, если B(S) было равно 0.

```
52          ENT1 0,3              G1      P ← R.
53          LD2  0,3(B)           G1      rI2 ← B(R).
54          J2N  A8L              G1      Переход к шагу A8 при rI2 = a.
55   A9L    LD1  0,3(RLINK)       H1      A9. Двукратный поворот.
56          LDX  0,1(LLINK)       H1      LINK(a,P ← LINK(−a,R))
57          STX  0,3(RLINK)       H1        → LINK(−a,R).
58          ST3  0,1(LLINK)       H1      LINK(a,P) ← R.
59          LD2  0,1(B)           H1      rI2 ← B(P).
60          LDX  T1,2             H1      −a, 0 или 0
61          STX  0,4(B)           H1        → B(S).
62          LDX  T2,2             H1      0, 0 или a
63          STX  0,3(B)           H1        → B(R).
64   A8L    LDX  0,1(RLINK)       G1      A8. Однократный поворот.
65          STX  0,4(LLINK)       G1      LINK(a,S) ← LINK(−a,P).
66          ST4  0,1(RLINK)       G1      LINK(−a,P) ← S.
67          JMP  8F               G1      Объединение с другой ветвью.
68   A7R    J2N  DONE             U2      Выход при rI2 = −a.
69          J2Z  6F               G2 + J2 Переход, если B(S) было равно 0.
70          ENT1 0,3              G2      P ← R.
71          LD2  0,3(B)           G2      rI2 ← B(R).
72          J2P  A8R              G2      Переход к шагу A8 при rI2 = a.
73   A9R    LD1  0,3(LLINK)       H2      A9. Двукратный поворот.
74          LDX  0,1(RLINK)       H2      LINK(a,P ← LINK(−a,R))
75          STX  0,3(LLINK)       H2        → LINK(−a,R).
76          ST3  0,1(RLINK)       H2      LINK(a,P) ← R.
77          LD2  0,1(B)           H2      rI2 ← B(P).
78          LDX  T2,2             H2      −a, 0 или 0
79          STX  0,4(B)           H2        → B(S).
80          LDX  T1,2             H2      0, 0 или a
81          STX  0,3(B)           H2        → B(R).
82   A8R    LDX  0,1(LLINK)       G2      A8. Однократный поворот.
83          STX  0,4(RLINK)       G2      LINK(a,S) ← LINK(−a,P).
84          ST4  0,1(LLINK)       G2      LINK(−a,P) ← S.
85   8H     STZ  0,1(B)           G       B(P) ← 0.
86   A10    CMP4 0,5(RLINK)       G       A10. Последний штрих.
87          JNE  *+3              G       Переход при RLINK(T) ≠ S.
88          ST1  0,5(RLINK)       G3      RLINK(T) ← P.
89          JMP  DONE             G3      Выход.
90          ST1  0,5(LLINK)       G4      LLINK(T) ← P.
91          JMP  DONE             G4      Выход.
92          CON  +1
93   T1     CON  0                        Таблица для (3).
94   T2     CON  0
95          CON  -1
96   6H     ENTX +1               J2      rX ← +1.
97   7H     STX  0,4(B)           J       B(S) ← a.
98          LDX  HEAD(LLINK)      J       LLINK(HEAD)
99          INCX 1                J         + 1
100         STX  HEAD(LLINK)      J           → LLINK(HEAD).
101  DONE   EQU  *                1 − S   Вставка завершена. ▌
```

Анализ вставки в сбалансированное дерево. [Те, кому математика кажется скучной и недостойной внимания, могут пропустить материал до (10).] Для выяснения времени работы алгоритма А необходимо ответить на следующие вопросы.

- Сколько сравнений будет выполнено во время поиска?
- Как далеко друг от друга будут находиться узлы S и Q? Иными словами, сколько корректировок будет проведено на шаге А6?
- Как часто будут выполняться однократные и двукратные повороты?

Несложно найти верхнюю границу времени работы программы, воспользовавшись теоремой А, однако с практической точки зрения нас, естественно, интересует среднее время работы. До сих пор нет теоретического вывода поведения алгоритма в среднем в связи с его сложностью, однако были получены некоторые интересные частные теоретические и эмпирические результаты.

В первую очередь, нас может заинтересовать количество B_{nh} сбалансированных бинарных деревьев с n внутренними узлами и высотой h. Для небольших h из соотношений

$$B_0(z) = 1, \qquad B_1(z) = z, \qquad B_{h+1}(z) = zB_h(z)\bigl(B_h(z) + 2B_{h-1}(z)\bigr) \tag{5}$$

нетрудно вычислить производящую функцию $B_h(z) = \sum_{n\geq 0} B_{nh} z^n$ (см. упр. 6). Таким образом,

$$\begin{aligned} B_2(z) &= 2z^2 + z^3, \\ B_3(z) &= \qquad 4z^4 + 6z^5 + 4z^6 + z^7, \\ B_4(z) &= \qquad\qquad 16z^7 + 32z^8 + 44z^9 + \cdots + 8z^{14} + z^{15}, \end{aligned}$$

и вообще, $B_h(z)$ при $h \geq 3$ имеет вид

$$2^{F_{h+1}-1} z^{F_{h+2}-1} + 2^{F_{h+1}-2} L_{h-1} z^{F_{h+2}} + \text{сложные члены} + 2^{h-1} z^{2^h-2} + z^{2^h-1}, \tag{6}$$

где $L_k = F_{k+1} + F_{k-1}$. (Эта формула обобщает теорему А.) Общее количество сбалансированных деревьев высотой h равно $B_h = B_h(1)$ и удовлетворяет рекуррентному соотношению

$$B_0 = B_1 = 1, \qquad B_{h+1} = B_h^2 + 2B_hB_{h-1}, \tag{7}$$

так что $B_2 = 3$, $B_3 = 3 \cdot 5$, $B_4 = 3^2 \cdot 5 \cdot 7$, $B_5 = 3^3 \cdot 5^2 \cdot 7 \cdot 23$; в общем случае

$$B_h = A_0^{F_h} A_1^{F_{h-1}} \ldots A_{h-1}^{F_1} A_h^{F_0}, \tag{8}$$

где $A_0 = 1$, $A_1 = 3$, $A_2 = 5$, $A_3 = 7$, $A_4 = 23$, $A_5 = 347$, $\ldots$, $A_h = A_{h-1}B_{h-2} + 2$. Последовательности B_h и A_h растут очень быстро — как *экспонента экспоненты*; в упр. 7 показано, что существует действительное число $\theta \approx 1.43687$, такое, что

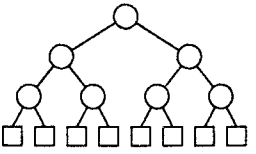
$$B_h = \lfloor \theta^{2^h} \rfloor - \lfloor \theta^{2^{h-1}} \rfloor + \lfloor \theta^{2^{h-2}} \rfloor - \cdots + (-1)^h \lfloor \theta^{2^0} \rfloor. \tag{9}$$

Если положить, что все B_h деревьев равновероятны, то, как показано в упр. 8, среднее число узлов в дереве высотой h равно

$$B_h'(1)/B_h(1) \approx (0.70118)2^h - 1. \tag{10}$$

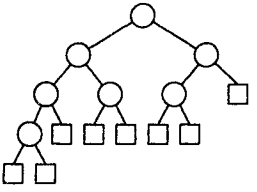
Это означает, что высота сбалансированного дерева с N узлами обычно гораздо ближе к $\log_2 N$, чем к $\log_\phi N$.

К сожалению, эти результаты в действительности не имеют отношения к алгоритму А, поскольку механизм построения деревьев делает одни из них существенно более вероятными, чем другие. Например, рассмотрим случай, когда $N = 7$, при котором имеется 17 сбалансированных деревьев. Существует $7! = 5\,040$ возможных способов вставки ключей в дерево. При этом идеально сбалансированное "совершенное" дерево



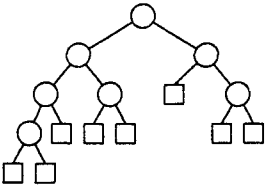
(11)

получается 2 160 раз. Дерево же Фибоначчи



(12)

образуется только в 144 случаях, а похожее на него дерево



(13)

будет получено 216 раз. Заменив левые поддеревья в (12) и (13) произвольными сбалансированными деревьями с четырьмя узлами и использовав зеркальное отображение относительно вертикальной оси, получим 16 различных деревьев. Восемь из них, полученные из (12), встречаются по 144 раза, а полученные из (13) — по 216 раз. Как это ни странно, деревья (13) встречаются чаще, чем (12).

Тот факт, что идеально сбалансированные деревья образуются с такой высокой вероятностью, и формула (10), соответствующая случаю равных вероятностей, делают весьма правдоподобным заключение о том, что для среднего времени поиска по сбалансированному дереву необходимо требовать около $\lg N + c$ сравнений, где c мало́. Р. В. Флойд (R. W. Floyd) обнаружил, что коэффициент при $\lg N$ не равен в точности 1, поскольку тогда корень дерева должен быть очень близок к медиане, а корни поддеревьев — около четвертых частей. Однако однократные и двукратные повороты не могут просто оставить корень дерева в медиане. Эмпирические исследования показали, что реальное среднее количество сравнений, необходимых для вставки N-го элемента, примерно равно $1.01 \lg N + 0.1$ при не очень малых N.

Для изучения поведения фаз вставки и балансировки алгоритма А можно классифицировать внешние узлы сбалансированного дерева, как показано на рис. 23. Путь, ведущий вверх из внешнего узла, определяется последовательностью плюсов и минусов ("+" для правой ссылки и "–" для левой ссылки). Сначала записываем последовательность до тех пор, пока не будет достигнут первый узел с ненулевым

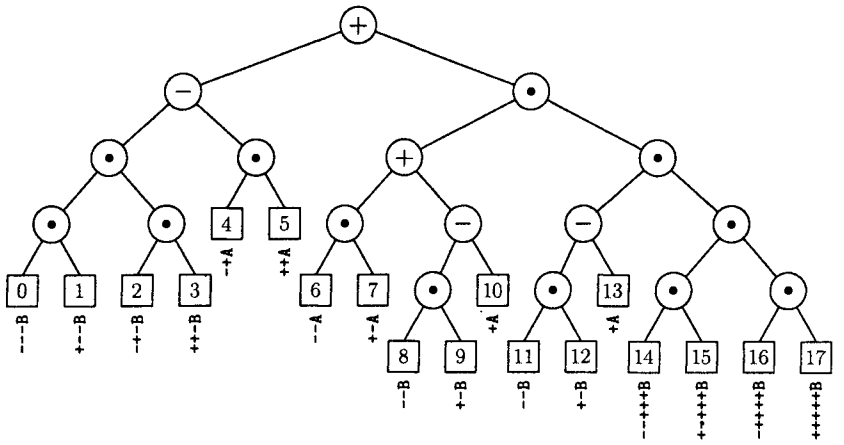


Рис. 23. Классификация, определяющая поведение алгоритма A после вставки.

фактором сбалансированности или (если такого узла нет) пока не будет достигнут корень. Затем добавляем A или B в соответствии с тем, будет ли новое дерево после вставки в данное место внутреннего узла сбалансированным. Так, путь вверх от $\boxed{3}$ записывается как ++-B, что означает "правая ссылка, правая ссылка, левая ссылка, несбалансировано". Запись, оканчивающаяся на A, означает, что балансировка после вставки нового узла не требуется, для записи, завершающейся на ++B или --B, необходим однократный поворот, а для записи, завершающейся на +-B или -+B, — двукратный поворот. Если в пути встречается k звеньев, на шаге A6 корректируется ровно $k-1$ факторов сбалансированности. Таким образом, описанные последовательности предоставляют необходимую информацию о времени работы шагов A6–A10.

Таблица 1

ПРИБЛИЖЕННЫЕ ВЕРОЯТНОСТИ ПРИ ВСТАВКЕ N-ГО ЭЛЕМЕНТА

Длина пути k	Нет корректировки	Однократный поворот	Двукратный поворот
1	.143	.000	.000
2	.152	.143	.143
3	.092	.048	.048
4	.060	.024	.024
5	.036	.010	.010
> 5	.051	.009	.008
В среднем 2.78	Всего .534	.233	.232

Эмпирические тесты со случайными числами в количестве $100 \le N \le 2000$ дали приближенные вероятности для путей различных типов (табл. 1); очевидно, эти вероятности быстро сходятся к предельным значениям при $N \to \infty$. В табл. 2 приведены точные значения вероятностей, соответствующих приведенным в табл. 1 (при $N = 10$), в предположении, что все 10! перестановок равновероятны. (Вероятность, показанная в табл. 1 как имеющая значение .143, в действительности равна

1/7 для всех $N \geq 7$; см. упр. 11.) Однократный и двукратный повороты практически равновероятны при $N \leq 15$, однако при $N \geq 16$ двукратные повороты встречаются немного реже.

Таблица 2

ТОЧНЫЕ ВЕРОЯТНОСТИ ПРИ ВСТАВКЕ 10-ГО ЭЛЕМЕНТА

Длина пути k	Нет корректировки	Однократный поворот	Двукратный поворот
1	1/7	0	0
2	6/35	1/7	1/7
3	4/21	2/35	2/35
4	0	1/21	1/21
В среднем 247/105	53/105	26/105	26/105

Из табл. 1 видно, что $k \leq 2$ с вероятностью около $.143 + .152 + .143 + .143 = .581$; следовательно, шаг A6 почти в 60% случаев тривиален. Среднее количество изменений фактора сбалансированности с 0 на ± 1 на этом шаге примерно равно 1.8. Среднее количество изменений фактора сбалансированности с ± 1 до 0 на шагах A7–A10 составляет примерно $.534 + 2(.233 + .232) \approx 1.5$. Таким образом, вставив новый узел, можно добавить в среднем около $1.8 - 1.5 = 0.3$ несбалансированного узла. Это согласуется с тем, что около 68% всех узлов в сбалансированных случайных деревьях, построенных по алгоритму A, оказываются сбалансированными.

Приближенная модель алгоритма A была предложена К. К. Фостером (C. C. Foster) [*Proc. ACM Nat. Conf.* **20** (1965), 192–205]. Эту модель сложно назвать абсолютно точной, но она достаточно близка к истине, чтобы дать некоторое понимание происходящего. Предположим, что в большом дереве, построенном по алгоритму A, фактор сбалансированности узла соответственно равен 0 с вероятностью p, равен -1 с вероятностью $\frac{1}{2}(1-p)$ и равен $+1$ с той же вероятностью. Предположим далее (без обоснования), что факторы сбалансированности различных узлов независимы. Значит, вероятность того, что на шаге A6 число ненулевых факторов $\nu - 1$ равно $k-1$, равна $p^{k-1}(1-p)$, так что среднее значение ν равно $1/(1-p)$. Вероятность того, что потребуется повернуть часть дерева, равна $q \approx \frac{1}{2}$. Вставка нового узла должна увеличить количество сбалансированных узлов в среднем на p; это число увеличивается на 1 на шаге A5, на $-p/(1-p)$ на шаге A6, на q на шаге A7 и на $2q$ на шаге A8 или A9, так что должно получиться

$$p = 1 - p/(1-p) + 3q \approx 5/2 - p/(1-p).$$

Решение этого уравнения относительно p дает хорошее согласование с табл. 1:

$$p \approx \frac{9-\sqrt{41}}{4} \approx 0.649; \qquad 1/(1-p) \approx 2.851. \tag{14}$$

Время работы фазы поиска программы A (строки 01–19) равно

$$10C + C1 + 2D + 2 - 3S, \tag{15}$$

где C, $C1$ и S те же, что и в предыдущих алгоритмах этой главы, а D — число несбалансированных узлов, которые проходятся при поиске. Эмпирические тесты

показывают, что можно принять $D \approx \frac{1}{3}C$, $C1 \approx \frac{1}{2}(C+S)$, $C+S \approx 1.01 \lg N + 0.1$, так что среднее время поиска примерно равно $11.3 \lg N + 3.3 - 13.7S$ единиц. (Если поиск осуществляется значительно чаще вставки, конечно же, можно использовать отдельную, более быструю программу поиска, поскольку не нужно следить за факторами сбалансированности. В этом случае среднее время работы при успешном поиске составило бы около $(6.6 \lg N - 3.4)u$ и даже в наихудшем случае было бы меньше, чем среднее время работы программы 6.2.2T.)

При неудачно завершенном поиске время работы фазы вставки программы A (строки 20–45) составляет $8F + 26 + (0, 1$ или $2)$ единиц. Данные из табл. 1 показывают, что в среднем $\acute{F} \approx 1.8$. Для фазы балансировки (строки 46–101) требуется 16.5, 8, 27.5 или 45.5 (± 0.5) единиц времени в зависимости от того, что мы делаем: увеличиваем общую высоту, просто выходим (без балансировки) или выполняем однократный либо двукратный поворот. Первый случай практически не встречается; вероятности остальных составляют около .534, .233 и .232, так что среднее время работы комбинированной "вставочно-балансировочной" части программы A — примерно $63u$.

Эти числа показывают, что операции над сбалансированными деревьями выполняются достаточно быстро, хотя программы при этом несколько больше по размеру. При случайных входных данных простой алгоритм вставки в дерево, описанный в разделе 5.2.2, работает быстрее примерно на $50u$ на одну вставку. Однако, используя сбалансированные деревья, можно получить хорошие результаты даже при неслучайных входных данных.

Один из способов сравнения программы A с программой 6.2.2T заключается в рассмотрении наихудшего для последней программы случая. Если попытаться выяснить, сколько времени потребуется для вставки N ключей в порядке возрастания в изначально пустое дерево, то окажется, что программа A работает медленнее при $N \leq 26$ и быстрее — при $N \geq 27$.

Представление линейных списков. Теперь вернемся к следующему сделанному в начале этого раздела замечанию: сбалансированные деревья могут использоваться для представления линейных списков таким образом, что можно будет быстро вставлять элементы в список, преодолевая трудности, которые связаны с последовательным расположением элементов, и обеспечивая при этом произвольный доступ к элементам списка, т. е. преодолевая сложности связанного размещения элементов.

Идея состоит во введении нового поля RANK в каждом узле. Это поле указывает относительное положение узла в его поддереве, а именно — единица плюс количество узлов в его левом поддереве. На рис. 24 показаны значения RANK для бинарного дерева, приведенного на рис. 23. При представлении списков поле KEY можно полностью исключить (при желании можно оставить оба поля, чтобы иметь возможность находить элементы как по значению ключа, так и по относительному положению в списке).

Используя такое поле RANK, можно свести поиск по положению элемента к модификации уже изученных нами алгоритмов.

Алгоритм B (*Поиск в дереве по положению элемента*). Дан линейный список, представленный в виде бинарного дерева. Алгоритм позволяет найти k-й элемент списка (k-й узел дерева в симметричном порядке) по заданному k. Предполагается,

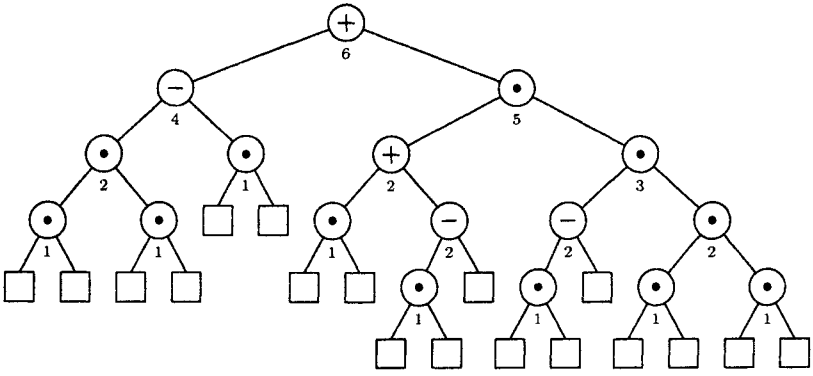


Рис. 24. Поля RANK для поиска по положению элемента в списке.

что, как и в алгоритме A, имеется головной узел и что узлы дерева имеют поля LLINK и RLINK, а также описанное поле RANK.

B1. [Инициализация.] Установить M $\leftarrow k$, P $\leftarrow$ RLINK(HEAD).

B2. [Сравнение.] Если P $= \Lambda$, алгоритм заканчивается неудачно (это может произойти, только если k было больше, чем количество узлов в дереве, или $k \leq 0$). В противном случае, если M $<$ RANK(P), перейти к шагу B3; если M $>$ RANK(P), перейти к шагу B4; если M $=$ RANK(P), алгоритм успешно завершается (P указывает на k-й узел).

B3. [Перемещение влево.] Присвоить P $\leftarrow$ LLINK(P) и вернуться к шагу B2.

B4. [Перемещение вправо.] Присвоить M $\leftarrow$ M $-$ RANK(P) и P $\leftarrow$ RLINK(P) и вернуться к шагу B2. ▌

В этом алгоритме определенный интерес представляет только операция над M на шаге B4. Аналогично можно модифицировать процедуру вставки элемента, хотя в этом случае имеются определенные тонкости.

Алгоритм C (*Вставка в сбалансированное дерево по положению*). Дан линейный список, представленный в виде сбалансированного бинарного дерева. Алгоритм вставляет новый узел непосредственно перед k-м элементом списка по заданным k и указателем Q на новый узел. Если $k = N + 1$, новый узел вставляется за последним элементом списка.

Бинарное дерево, как и в случае использования алгоритма A, предполагается непустым, имеющим головной узел; также предполагается, что узлы имеют поля LLINK, RLINK и B, а также поле RANK, описанное выше. Этот алгоритм очень похож на алгоритм A; отличие заключается в использовании и обновлении полей RANK вместо KEY.

C1. [Инициализация.] Установить T $\leftarrow$ HEAD, S $\leftarrow$ P $\leftarrow$ RLINK(HEAD), U $\leftarrow$ M $\leftarrow k$.

C2. [Сравнение.] Если M $\leq$ RANK(P), перейти к шагу C3, в противном случае перейти к шагу C4.

C3. [Перемещение влево.] Установить RANK(P) $\leftarrow$ RANK(P) $+ 1$ (будем вставлять новый узел слева от P). Установить R $\leftarrow$ LLINK(P). Если R $= \Lambda$, присвоить LLINK(P) $\leftarrow$ Q и перейти к шагу C5. В противном случае, если B(R) $\neq 0$,

присвоить T ← P, S ← R и U ← M. И, наконец, присвоить P ← R и вернуться к шагу C2.

C4. [Перемещение вправо.] Установить M ← M − RANK(P) и R ← RLINK(P). Если R = Λ, присвоить RLINK(P) ← Q и перейти к шагу C5. В противном случае, если B(R) ≠ 0, присвоить T ← P, S ← R и U ← M. И, наконец, присвоить P ← R и вернуться к шагу C2.

C5. [Вставка.] Установить RANK(Q) ← 1, LLINK(Q) ← RLINK(Q) ← Λ, B(Q) ← 0.

C6. [Корректировка факторов сбалансированности.] Установить M ← U. (Это действие позволяет восстановить предыдущее значение M, когда P было равно S; все поля RANK установлены соответствующим образом.) Если M < RANK(S), присвоить R ← P ← LLINK(S) и a ← −1; в противном случае присвоить R ← P ← RLINK(S), a ← +1 и M ← M − RANK(S). Затем повторять следующие действия, пока P не станет равным Q: если M < RANK(P), установить B(P) ← −1 и P ← LLINK(P); если M > RANK(P), установить B(P) ← +1, M ← M − RANK(P) и P ← RLINK(P). (Если M = RANK(P), то P = Q и можно переходить к следующему шагу.)

C7. [Балансировка.] Здесь возможны несколько случаев.

i) Если B(S) = 0, установить B(S) ← a, LLINK(HEAD) ← LLINK(HEAD) + 1 и прекратить выполнение алгоритма.

ii) Если B(S) = −a, установить B(S) ← 0 и прекратить выполнение алгоритма.

iii) Если B(S) = a, то перейти к шагу C8 в случае, если B(R) = a, и к шагу C9, если B(R) = −a.

C8. [Однократный поворот.] Установить P = R, LINK(a,S) ← LINK(−a,R), LINK(−a, R) ← S, B(S) ← B(R) ← 0. Если a = +1, установить RANK(R) ← RANK(R) + RANK(S); если a = −1, установить RANK(S) ← RANK(S) − RANK(R). Перейти к шагу C10.

C9. [Двукратный поворот.] Выполнить шаг A9 (алгоритм A). Затем, если a = +1, установить RANK(R) ← RANK(R) − RANK(P), RANK(P) ← RANK(P) + RANK(S); если a = −1, присвоить RANK(P) ← RANK(P)+RANK(R), а затем RANK(S) ← RANK(S) − RANK(P).

C10. [Последний штрих.] Если S = RLINK(T), присвоить RLINK(T) ← P, в противном случае — LLINK(T) ← P. ▮

***Удаление, конкатенация и другие операции.** Существует множество других операций над сбалансированными деревьями, которые поддерживают их сбалансированность, однако их алгоритмы слишком длинны для детального рассмотрения в этой книге. Здесь обсуждаются только общие идеи, и заинтересованному читателю предоставляется возможность самостоятельно восстановить опущенные детали (что не так трудно, как кажется на первый взгляд).

Удаление при корректной постановке выполняется за $O(\log N)$ шагов [C. C. Foster, "A Study of AVL Trees", Goodyear Aerospace Corp. report GER-12158 (April, 1965)]. Прежде всего, удаление произвольного узла сводится к простому удалению узла P, поля LLINK(P) или RLINK(P) которого равны Λ, как в алгоритме 6.2.2D.

Алгоритм к тому же должен быть изменен таким образом, чтобы он строил список указателей, определяющих путь к узлу P:

$$(P_0, a_0), \qquad (P_1, a_1), \qquad \ldots, \qquad (P_l, a_l), \tag{16}$$

где $P_0 = \texttt{HEAD}$, $a_0 = +1$; $\texttt{LINK}(a_i, P_i) = P_{i+1}$, $0 \le i < l$; $P_l = \texttt{P}$ и $\texttt{LINK}(a_l, P_l) = \Lambda$. Этот список при поиске вниз по дереву может быть помещен во вспомогательный стек. В процессе удаления узла P устанавливается $\texttt{LINK}(a_{l-1}, P_{l-1}) \leftarrow \texttt{LINK}(-a_l, P_l)$ и следует откорректировать фактор сбалансированности узла P_{l-1}. Предположим, что мы должны откорректировать фактор сбалансированности узла P_k, потому что уменьшилась высота поддерева a_k данного узла. Для этого используется следующая процедура: если $k = 0$, установить $\texttt{LLINK(HEAD)} \leftarrow \texttt{LLINK(HEAD)} - 1$ и прекратить выполнение алгоритма, поскольку уменьшилась высота всего дерева. В противном случае рассмотрим фактор сбалансированности $\texttt{B}(P_k)$. Возможны три варианта, приведенных ниже.

i) $\texttt{B}(P_k) = a_k$. Установить $\texttt{B}(P_k) \leftarrow 0$, уменьшить k на 1 и повторить процедуру корректировки для нового значения k.

ii) $\texttt{B}(P_k) = 0$. Установить $\texttt{B}(P_k) \leftarrow -a_k$ и прекратить выполнение алгоритма.

iii) $\texttt{B}(P_k) = -a_k$. Требуется балансировка!

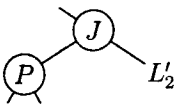
Ситуации, в которых требуется ребалансировка дерева, практически те же, что в алгоритме вставки; вернитесь к (1), где в роли A выступает узел P_k, а в роли B — узел $\texttt{LINK}(-a_k, P_k)$, принадлежащий ветви, *противоположной* той, в которой производится удаление. Отличие заключается в том, что узел B может быть сбалансированным. Это приводит к возникновению нового случая 3, который подобен случаю 1, но высота β равна $h + 1$. В случаях 1 и 2 ребалансировка, как и в (2), означает, что мы уменьшаем высоту, устанавливая $\texttt{LINK}(a_{k-1}, P_{k-1})$ в качестве корня (2), уменьшаем k на 1 и перезапускаем процедуру корректировки для нового значения k. В случае 3 выполняется однократный поворот, который оставляет факторы сбалансированности A и B ненулевыми без изменения общей высоты. После занесения в $\texttt{LINK}(a_{k-1}, P_{k-1})$ адреса узла B алгоритм завершается.

Важное отличие между удалением и вставкой заключается в том, что для удаления может потребоваться до $\log N$ поворотов, в то время как для вставки никогда не требуется больше одного. Причина этого становится ясна, если попытаться удалить крайний справа узел дерева Фибоначчи (см. рис. 8 в разделе 6.2.1). Однако эмпирические тесты показывают, что в среднем на одно удаление приходится около 0.21 вращений.

При использовании сбалансированных деревьев для представления линейных списков необходим алгоритм *конкатенации* (слияния); при этом некоторое дерево L_2 целиком вставляется справа от дерева L_1 без нарушения сбалансированности. Элегантный алгоритм конкатенации впервые был разработан Кларком А. Крейном (Clark A. Crane). Предположим, что высота$(L_1) \ge$ высота(L_2) (другой случай обрабатывается аналогично). Удалим из L_2 первый узел, назвав его *стыковочным узлом* J. Через L_2' обозначим получившееся дерево $L_2 \setminus \{J\}$. После этого спустимся по правым деревьям L_1, пока не достигнем узла P, такого, что

$$\text{высота}(P) - \text{высота}(L_2') = 0 \text{ или } 1.$$

Это всегда возможно, поскольку при переходе вниз на один уровень высота изменяется на 1 или 2. Теперь заменим на



и продолжим корректировку L_1, как если бы новый узел J был только что вставлен при помощи алгоритма А.

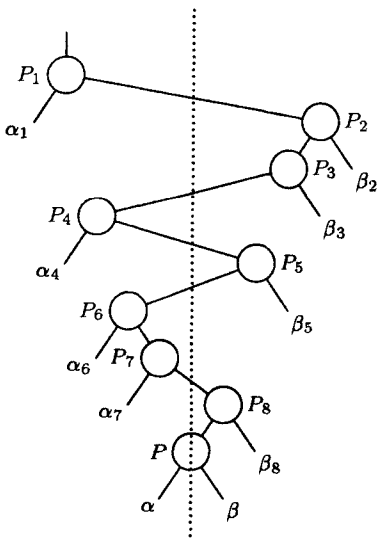


Рис. 25. Задача разделения списка.

Крейн также решил более сложную обратную задачу — *разделить* список на две части, которые дадут исходное дерево при конкатенации. Рассмотрим, например, задачу разделения списка, показанного на рис. 20, для получения двух списков, в одном из которых содержится $\{\mathtt{A},\ldots,\mathtt{I}\}$, а в другом — $\{\mathtt{J},\ldots,\mathtt{Q}\}$. При этом требуется существенная перекомпоновка деревьев. В общем случае, когда необходимо разделить дерево в некотором заданном узле P, путь к P будет похож на путь, изображенный на рис. 25. Нужно построить левое дерево, содержащее узлы $\alpha_1, P_1, \alpha_4, P_4, \alpha_6, P_6, \alpha_7, P_7, \alpha, P$ в симметричном порядке, и правое дерево, которое содержит $\beta, P_8, \beta_8, P_5, \beta_5, P_3, \beta_3, P_2, \beta_2$. Это можно сделать с помощью последовательности конкатенаций: сначала вставить P справа от α, затем соединить β с β_8 с использованием P_8 в качестве стыковочного узла, соединить α_7 с αP (стыковочный узел — P_7), α_6 с $\alpha_7 P_7 \alpha P$ (стыковочный узел — P_6), $\beta P_8 \beta_8$ с β_5 (стыковочный узел — P_5) и т. д. Узлы $P_8, P_7, \ldots, P_1$ на пути к P используются в качестве стыковочных. Крейн доказал, что для такого алгоритма разделения требуется $O(\log N)$ единиц времени для исходного дерева с N узлами, так как конкатенация с использованием данного стыковочного узла выполняется за $O(k)$ шагов, где k —

разность высот конкатенируемых деревьев, а значения k образуют геометрическую прогрессию.

Все эти алгоритмы могут использоваться как с полями KEY, так и с полями RANK (или с обоими), хотя в случае конкатенации с помощью ключей все ключи дерева L_2 должны быть больше ключей дерева L_1. Для общих целей часто предпочтительнее использовать деревья с *тремя связями*, в которых наряду с полями LLINK и RLINK применяется поле UP, которое указывает на родительский узел, и однобитовое поле, указывающее, каким потомком является данный узел: левым или правым. Такие деревья упрощают используемые алгоритмы и позволяют определить узел без явного указания пути к нему. Можно написать подпрограмму для удаления узла NODE(P) по заданному P, удаления узла, следующего за NODE(P) в симметричном порядке, для нахождения списка, содержащего NODE(P), и т. д. В алгоритме удаления для деревьев с тремя связями не нужно строить список (16), так как ссылки UP предоставляют всю необходимую информацию. Конечно, при вставке, удалении или повороте в таких деревьях требуется немного больше усилий для изменения ссылок. Использование "трехсвязных" деревьев вместо двухсвязных аналогично использованию двойных связей вместо одинарных: можно начать работу с любого места и двигаться как вперед, так и назад. Полное описание алгоритмов, основанных на трехсвязных деревьях, приводится в диссертации Крейна (Stanford University, 1972).

Альтернативы AVL-деревьям. Помимо использования AVL-деревьев, было предложено несколько других подходов к организации деревьев, гарантирующих логарифмическое время доступа. Например, К. К. Фостер (C. C. Foster) [*CACM* **16** (1973), 513–517] рассмотрел бинарные деревья, которые получаются, если ограничить разность высот поддеревьев некоторой величиной k. Такие структуры называются HB(k) (что означает "height-balanced" — "сбалансированные по высоте"). В этих терминах рассмотренные сбалансированные деревья представляют собой частный случай HB(1).

Интересная концепция *взвешенно-сбалансированных деревьев* (*weight-balanced trees*) была изучена Ю. Нивергельтом (J. Nievergelt), Э. Рейнгольдом (E. Reingold) и Ч. К. Вонгом (C. K. Wong). Они не рассматривали высоту деревьев, а поставили условие, которому должны удовлетворять поддеревья всех узлов:

$$\sqrt{2}-1 < \frac{\text{левый вес}}{\text{правый вес}} < \sqrt{2}+1, \tag{17}$$

где левый и правый веса означают количество *внешних* узлов в левом и правом поддеревьях соответственно. Можно показать, что при вставке взвешенную сбалансированность можно поддерживать с помощью однократных и двукратных поворотов, как в алгоритме A (см. упр. 25). Однако при вставке могут оказаться необходимыми несколько ребалансировок дерева. При ослаблении условия (17) количество нужных ребалансировок уменьшается (правда, за счет увеличения времени поиска).

На первый взгляд может показаться, что для взвешенно-сбалансированных деревьев необходимо больше памяти, однако иногда ее нужно даже меньше, чем для обычных сбалансированных деревьев! Если в каждом узле уже имеется поле RANK для представления линейного списка, то это и есть вес левого поддерева, а соответствующие правые веса могут быть определены при движении вниз по дереву.

Однако требуемые для сохранения взвешенной сбалансированности накладные расходы отнимают больше времени, чем алгоритм A, и сохранение двух битов на узле не представляется достаточной компенсацией перерасхода времени.

Почему вы не сдвоите их в тройки?

(*Why don't you pair 'em up in threes?*)

— Приписывается ЁДЖИ БЕРРА (YOGI BERRA) (1970)

Еще один интересный подход к AVL-деревьям, называемый "2-3-деревья", был предложен Джоном Хопкрофтом (John Hopcroft) в 1970 году [см. Aho, Hopcroft, Ullman, *The Design and Analysis of Computer Algorithms* (Reading, Mass.: Addison-Wesley, 1974), Chapter 4]. Идея этого подхода заключается в том, что из каждого узла могут выходить либо две, либо три ветви; при этом требуется, чтобы все внешние узлы находились на одном уровне. Каждый внутренний узел содержит либо один, либо два ключа, как показано на рис. 26.

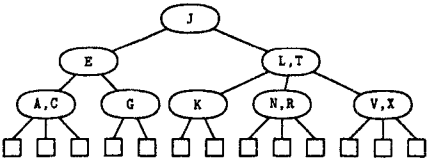


Рис. 26. 2-3-дерево.

Вставку в 2-3-дерево пояснить несколько проще по сравнению со вставкой в AVL-дерево: чтобы поместить новый ключ в узел, в котором содержится ровно один ключ, необходимо просто вставить его как второй ключ. С другой стороны, если в узле уже содержатся два ключа, делим их на два "одноключевых" узла и вставляем средний ключ в родительский узел. Это, в свою очередь, может привести к делению родительского узла. На рис. 27 показан описанный процесс вставки ключа в 2-3-дерево, изображенное на рис. 26.

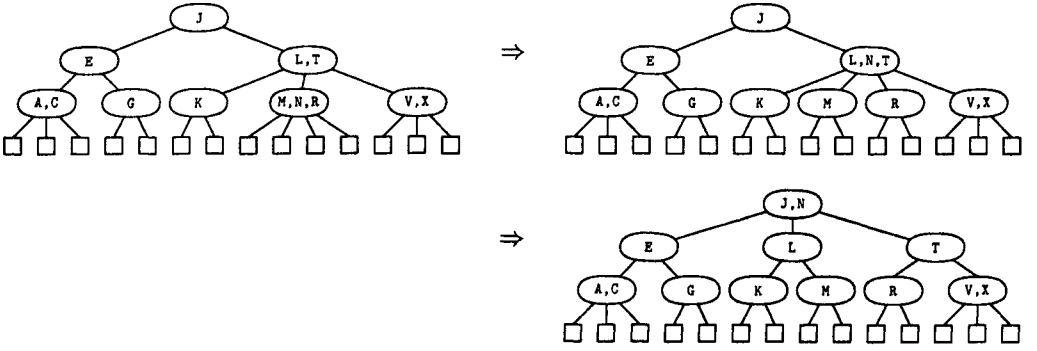


Рис. 27. Вставка нового ключа M в 2-3-дерево, приведенное на рис. 26.

Дж. Э. Хопкрофт (J. E. Hopcroft) обнаружил, что удаление, конкатенация и разделение с 2-3-деревьями проводятся достаточно просто и очень похожи на аналогичные операции с AVL-деревьями.

Р. Байер (R. Bayer) [*Proc. ACM-SIGFIDET Workshop* (1971), 219–235] предложил интересное представление 2-3-деревьев в виде бинарных деревьев. Взгляните на рис. 28, на котором показано такое представление дерева, изображенного на рис. 26. Один бит каждого узла используется для того, чтобы отличать "горизонтальные" правые ссылки RLINK от "вертикальных". Заметьте, что ключи в таком дереве расположены, как и в бинарном дереве поиска, в симметричном порядке слева направо. Оказывается, что при вставке нового ключа в деревья такого типа необходимо выполнить те же преобразования, что и при вставке в бинарное дерево, а именно — однократные и двукратные повороты (хотя в этом случае требуется только одна версия каждого поворота без отражений относительно вертикальной оси, необходимых для алгоритмов A и C).

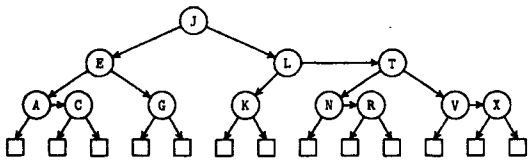


Рис. 28. 2-3-дерево (см. рис. 26), представленное в виде бинарного дерева поиска.

Развитие этих идей привело к появлению различных вариантов сбалансированных деревьев, среди которых особенно известны *красно-черные деревья* (*red-black trees*), именуемые также симметричными бинарными B-деревьями или полусбалансированными деревьями [R. Bayer, *Acta Informatica* **1** (1972), 290–306; L. Guibas and R. Sedgewick, *FOCS* **19** (1978), 8–21; H. J. Olivié, *RAIRO Informatique Théorique* **16** (1982), 51–71; R. E. Tarjan, *Inf. Proc. Letters* **16** (1983), 253–257; T. H. Cormen, C. E. Leiserson and R. L. Rivest, *Introduction to Algorithms* (MIT Press, 1989), Chapter 14; R. Sedgewick, *Algorithms in C* (Addison-Wesley, 1997), §13.4]. Существует, кроме того, связанное семейство деревьев, называемых истерическими B-деревьями (hysterical B-trees) или (a, b)-деревьями, в частности $(2, 4)$-деревья [D. Maier and S. C. Salveter, *Inf. Proc. Letters* **12** (1981), 199–202; S. Huddleston and K. Mehlhorn, *Acta Informatica* **17** (1982), 157–184].

Когда одни ключи встречаются чаще других, логично разместить более "частые" ключи ближе к корню дерева, как в случае использования оптимальных бинарных деревьев, о которых шла речь в разделе 6.2.2. Динамические деревья, которые делают возможным поддержание взвешенной сбалансированности в заданных пределах около оптимальной, называемые *"смещенными" деревьями* (*biased trees*), были разработаны С. В. Бентом (S. W. Bent), Д. Д. Слитором (D. D. Sleator) и Р. Е. Таржаном (R. E. Tarjan) [*SICOMP* **14** (1985), 545–568; J. Feigenbaum and R. E. Tarjan, *Bell System Tech. J.* **62** (1983), 3139–3158]. К сожалению, соответствующие алгоритмы слишком сложны.

Существенно проще самокорректирующиеся структуры данных, называемые *вытянутыми деревьями* (*splay tree*), которые были разработаны Д. Д. Слитором и Р. Е. Таржаном [*JACM* **32** (1985), 652–686]. Они основаны на идеях, подобных используемым при эвристических перемещениях и перестановках (см. раздел 6.1). Данная технология была исследована Б. Алленом (B. Allen) и Дж. Мунро (J. Munro) [*JACM* **25** (1978), 526–535], а также Дж. Битнером (J. Bitner) [*SICOMP* **8** (1979),

82–110]. Вытянутые деревья, подобно другим видам уже упоминавшихся сбалансированных деревьев, поддерживают операции как вставки и удаления, так и конкатенации и разделения, причем достаточно простым образом. Более того, время, необходимое для доступа к данным в вытянутом дереве, не превышает времени доступа в статически оптимальном дереве, умноженного на некоторую небольшую константу (в установившемся режиме после серии операций с деревом). Слитор и Таржан предположили, что общее время доступа в вытянутом дереве не превышает оптимального времени доступа к данным и времени, необходимого для выполнения динамических поворотов согласно любому алгоритму работы с бинарными деревьями, умноженного на константу.

Рандомизация позволяет использовать методы, даже более быстрые и простые, чем вытянутые деревья. Жан Вуйлемен (Jean Vuillemin) [*CACM* **23** (1980), 229–239] ввел понятие *декартовых деревьев* (*cartesian trees*), в которых каждый узел имеет два ключа (x, y). Часть x упорядочена слева направо, как в бинарных деревьях поиска; часть y упорядочена сверху вниз, как в случае приоритетной очереди (см. раздел 5.2.3). С. Р. Арагон (C. R. Aragon) и Р. Г. Зайдель (R. G. Seidel) присвоили таким структурам данных очень цветастое название *treap*, которое получено в результате комбинирования частей слов *tree*s и he*ap*s. (Видимо, наилучший русский термин должен звучать как *дуча* — от *д*ерева и к*учи*. — *Прим. перев.*) Только одна дуча может быть построена на основе n данных пар ключей $(x_1, y_1), \ldots, (x_n, y_n)$, если все x и y различны. Один из путей получения дучи — вставка x согласно алгоритму 6.2.2T в соответствии с порядком y. Однако имеется еще один простой алгоритм для непосредственной вставки любых новых пар в дучу. Арагон и Зайдель обнаружили [*FOCS* **30** (1989), 540–546], что, если x представляют собой обычные ключи, а y выбираются случайным образом, можно быть уверенным, что дуча имеет вид случайного бинарного дерева поиска. В частности, дуча со случайными значениями y всегда будет сбалансирована, за исключением случая экспоненциально малой вероятности (см. упр. 5.2.2–42). Арагон и Зайдель также показали, что дучи могут быть легко "скошены", так что, например, ключ x с относительной частотой f будет располагаться вблизи корня в соответствии со своей частотой, если с ним будет ассоциирована величина $y = U^{1/f}$, где U — случайное число между 0 и 1. Дучи, в целом, предпочтительнее вытянутых деревьев, как показали некоторые проведенные Д. Э. Кнутом эксперименты, связанные с расчетами выпуклых оболочек [*JACM* **45** (1998), 288–323].

В следующее издание книги планируется добавить раздел 6.2.5, посвященный рандомизированным структурам данных. В нем будут рассмотрены списки с пропусками [W. Pugh, CACM **33** *(1990), 668–676], рандомизированные бинарные деревья поиска [S. Roura and C. Martínez, JACM* **45** *(1998), 288–323], а также упомянутые в этом разделе "дучи".*

УПРАЖНЕНИЯ

1. [*01*] Почему в случае 2 в (1) нельзя просто поменять местами левые поддеревья узлов A и B?

2. [*16*] Объясните, почему, если достигнут шаг A7 с `B(S)` = 0, дерево становится на один уровень выше.

▶ **3.** [*M25*] Докажите, что сбалансированное дерево с N внутренними узлами не может содержать более $(\phi-1)N \approx 0.61803N$ узлов с ненулевыми факторами сбалансированности.

4. [*M22*] Докажите или опровергните следующее утверждение: среди всех сбалансированных деревьев с $F_{h+1}-1$ внутренними узлами дерево Фибоначчи порядка h имеет наибольшую длину внутреннего пути.

▶ **5.** [*M25*] Докажите или опровергните следующее утверждение: если для последовательной вставки ключей $K_2, \ldots, K_N$ в дерево, изначально содержащее только ключ K_1 ($K_1 < K_2 < \cdots < K_N$), используется алгоритм A, то полученное дерево всегда *оптимально* (т. е. имеет минимальную длину внутреннего пути среди всех бинарных деревьев с N узлами).

6. [*M21*] Докажите, что (5) определяет производящую функцию для сбалансированных деревьев высотой h.

7. [*M27*] (Н. Дж. А. Слоан (N. J. A. Sloane) и А. В. Ахо (A. V. Aho).) Докажите замечательную формулу (9) для числа сбалансированных деревьев высотой h. (*Указание.* Положим, что $C_n = B_n + B_{n-1}$, и используем тот факт, что $\log(C_{n+1}/C_n^2)$ очень мало́ при больших n.)

8. [*M24*] (Л. А. Хиздер.) Покажите, что существует константа β, такая, что $B_h'(1)/B_h(1) = 2^h\beta - 1 + O(2^h/B_{h-1})$ при $h \to \infty$.

9. [*HM44*] Чему равно асимптотическое число сбалансированных бинарных деревьев с n внутренними узлами $\sum_{h\geq 0} B_{nh}$? Чему равна асимптотическая средняя высота $\sum_{h\geq 0} hB_{nh}/$ $/\sum_{h\geq 0} B_{nh}$?

▶ **10.** [*27*] (Р. К. Ричардс (R. C. Richards).) Покажите, что сбалансированное дерево может быть единственным образом построено по списку его факторов сбалансированности `B(1)B(2)...B(N)` в симметричном порядке.

11. [*M24*] (Марк Р. Браун (Mark R. Brown).) Докажите, что при $n \geq 6$ среднее количество внешних узлов каждого из типов `+A`, `-A`, `++B`, `+-B`, `-+B`, `--B` составляет в точности $(n+1)/14$ для случайных сбалансированных деревьев с n внутренними узлами, построенных по алгоритму A.

▶ **12.** [*24*] Чему равно максимально возможное время работы программы A при вставке восьмого узла в сбалансированное дерево? Чему равно минимальное время для этого случая?

13. [*05*] Почему поле `RANK` лучше использовать так, как указано в тексте, а не хранить индекс каждого узла в качестве его ключа (называя первый узел "1", второй — "2" и т. д.)?

14. [*11*] Можно ли адаптировать алгоритмы 6.2.2T и 6.2.2D для работы с линейными списками с использованием поля `RANK`, как это было сделано для алгоритмов работы со сбалансированными деревьями?

15. [*18*] (К. А. Крейн (C. A. Crane).) Предположим, что упорядоченный линейный список представлен в виде бинарного дерева с полями `KEY` и `RANK` в каждом узле. Разработайте алгоритм, который осуществляет поиск в дереве данного ключа K и определяет его положение в списке, т. е. находит число m, такое, что меньше K лишь $m-1$ ключей.

▶ **16.** [*20*] Нарисуйте сбалансированное дерево, которое получается после удаления узла `E` и корневого узла `F` из дерева, показанного на рис. 20, с использованием предложенного в тексте алгоритма.

▶ **17.** [*21*] Нарисуйте сбалансированное дерево, которое получается после конкатенации дерева Фибоначчи (12): (a) справа и (b) слева от дерева, показанного на рис. 20, с помощью предложенного в тексте алгоритма конкатенации.

18. [*22*] Нарисуйте сбалансированные деревья, которые получаются после разделения дерева, показанного на рис. 20, на две части ($\{\mathtt{A},\ldots,\mathtt{I}\}$ и $\{\mathtt{J},\ldots,\mathtt{Q}\}$) с использованием предложенного в тексте алгоритма.

▶ **19.** [*26*] Найдите способ преобразования данного сбалансированного дерева так, чтобы фактор сбалансированности в корне не был равным -1. При этом должен сохраняться симметричный порядок узлов и должно порождаться искомое сбалансированное дерево за $O(1)$ единиц времени независимо от количества узлов в исходном дереве.

20. [*40*] Рассмотрите идею использования ограниченного класса сбалансированных деревьев с факторами сбалансированности, равными 0 или $+1$ (в этом случае поле B может свестись к одному биту). Существует ли для таких деревьев процедура вставки с разумной эффективностью?

▶ **21.** [*30*] (*Идеальная балансировка.*) Разработайте алгоритм построения бинарных деревьев с N узлами, которые оптимальны в смысле упр. 5. Он должен выполняться за $O(N)$ шагов и быть "последовательным", т. е. вы должны получать узлы по одному в порядке возрастания и строить частичные деревья по ходу, не зная окончательного значения N (такой алгоритм может пригодиться для перестройки плохо сбалансированного дерева или при слиянии двух деревьев в одно).

22. [*M20*] Каков аналог теоремы A для взвешенно-сбалансированных деревьев?

23. [*M20*] (Э. Рейнгольд (E. Reingold).) Покажите, что между сбалансированными по высоте и взвешенно-сбалансированными деревьями не существует простой взаимосвязи.

a) Докажите, что существуют сбалансированные по высоте деревья со сколь угодно малым отношением (вес левого поддерева)/(вес правого поддерева) в смысле (17).

b) Докажите, что существуют взвешенно-сбалансированные деревья, имеющие сколь угодно большую разность между высотой левого и правого поддеревьев.

24. [*M22*] (Э. Рейнгольд.) Докажите, что если усилить условие (17) до

$$\frac{1}{2} < \frac{\text{левый вес}}{\text{правый вес}} < 2,$$

то ему будут удовлетворять только идеально сбалансированные бинарные деревья с $2^n - 1$ внутренними узлами. (В таких деревьях левые и правые веса в каждом узле равны между собой.)

25. [*27*] (Ю. Нивергельт (J. Nievergelt), Э. Рейнгольд и Ч. Вонг (C. Wong).) Покажите, что можно разработать алгоритм вставки во взвешенно-сбалансированные деревья с сохранением условия (17), выполняющий не более $O(\log N)$ поворотов на вставку.

26. [*40*] Исследуйте свойства сбалансированных t-арных деревьев при $t > 2$.

▶ **27.** [*M23*] Оцените максимальное количество сравнений, необходимых для поиска в 2-3-дереве с N внутренними узлами.

28. [*41*] Реализуйте алгоритмы работы с 2-3-деревьями программно с достаточной эффективностью.

29. [*M47*] Проанализируйте поведение 2-3-деревьев в среднем при случайных вставках.

30. [*26*] (Э. Мак-Крейт (E. McCreight).) В разделе 2.5 обсуждался ряд стратегий динамического распределения памяти, включая метод наилучшего подходящего (best-fit) (выбор области наименьшего размера, удовлетворяющей запросу) и метод первого подходящего (first-fit) (выбор области с наименьшим адресом, удовлетворяющей запросу). Покажите, что если свободное пространство связано в сбалансированное дерево, то можно найти как (a) наилучшую подходящую, так и (b) первую подходящую области за $O(\log n)$ единиц времени, где n — количество доступных областей (алгоритмы из раздела 2.5 выполняют поставленную задачу за $O(n)$ шагов).

31. [*34*] (М. Л. Фредман (M. L. Fredman), 1975.) Придумайте представление линейных списков, такое, что вставка нового элемента между позициями $m-1$ и m при данном m требует $O(\log m)$ единиц времени.

32. [*M27*] Даны два бинарных дерева с n узлами — T и T'. Будем говорить, что $T \preceq T'$, если T' может быть получено из T с помощью последовательности из нуля или нескольких поворотов вправо. Докажите, что $T \preceq T'$ тогда и только тогда, когда $r_k \le r'_k$ для $1 \le k \le n$, где r_k и r'_k означают соответственно размеры правых поддеревьев k-х узлов (в симметричном порядке) деревьев T и T'.

▶ **33.** [*25*] (А. Л. Бухсбаум (A. L. Buchsbaum).) Поясните, каким образом можно неявно закодировать фактор сбалансированности AVL-дерева для сохранения двух битов на узел за счет дополнительной работы при обращении к дереву.

И велел Самуил подходить всем коленам
Израилевым, и указано колено Вениаминово.
И велел подходить колену Вениаминову
по племенам его, и указано племя Матриево;
и приводят племя Матриево по мужам,
и назван Саул, сын Кисов; и искали его,
и не находили.

— Первая книга Царств, 10:20–21

6.2.4. Сильноветвящиеся деревья

Обсуждавшиеся выше методы поиска были разработаны, в первую очередь, для внутреннего поиска, когда выполняется просмотр таблицы, целиком содержащейся в высокоскоростной внутренней памяти. Теперь рассмотрим *внешний* поиск, когда нужно получить информацию из очень большого файла, расположенного на внешних запоминающих устройствах с прямым доступом, таких как диски и барабаны (о дисках и барабанах можно прочесть в разделе 5.4.9).

Древовидные структуры довольно удобны для внешнего поиска, если выбрать подходящий способ представления дерева. Рассмотрим большое бинарное дерево поиска, показанное на рис. 29, и представим, что оно хранится в дисковом файле (поля `LLINK` и `RLINK` теперь представляют собой адреса на диске, а не во внутренней памяти). Если наивно проводить поиск в таком дереве так же, как в случае внутреннего поиска, нам понадобится около $\lg N$ обращений к диску для выполнения одного поиска. При N, равном миллиону, это означает порядка 20 обращений к диску. Однако стоит разделить таблицу на 7-узельные "страницы", как показано на рис. 29, и при доступе к одной странице за один раз потребуется примерно в три раза меньше обращений к диску, т. е. практически поиск ускорится в три раза!

Группирование узлов в страницы фактически приводит к преобразованию бинарного дерева в "октарное" с разветвлением в каждой странице-узле на 8 путей. Если страницы будут иметь бóльшие размеры, с разветвлением на 128 путей после каждого обращения к диску, то поиск элемента в таблице с миллионом элементов завершится при просмотре всего лишь трех страниц. Можно постоянно хранить корневую страницу во внутренней памяти, так что потребуются лишь два обращения к диску (при этом во внутренней памяти одновременно будет находиться не более 254 ключей).

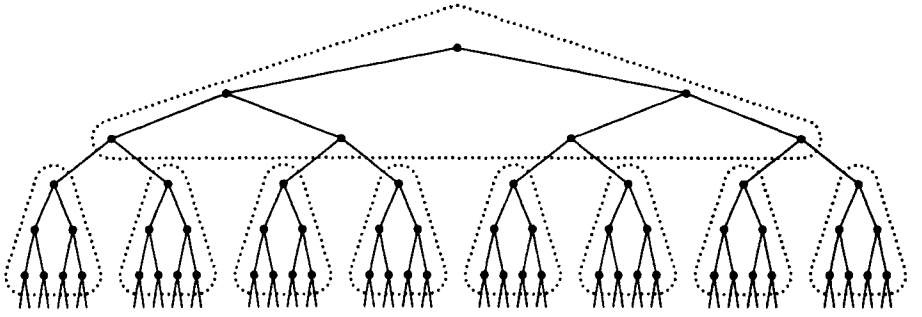


Рис. 29. Большое бинарное дерево поиска может быть разделено на "страницы".

Разумеется, не следует делать страницы очень большими, так как размеры внутренней памяти ограничены и чтение бóльшей страницы отнимает больше времени. Например, предположим, что для чтения страницы, допускающей разветвление на m путей, необходимо $72.5+0.05m$ ms. Время на внутреннюю обработку каждой страницы составит около $a + b \lg m$ ms, где a малó по сравнению с 72.5 ms, так что общее время поиска в большой таблице примерно пропорционально $\lg N$, умноженному на

$$(72.5 + 0.05m)/\lg m + b.$$

Эта величина достигает минимума при $m \approx 307$; в действительности этот минимум очень "широк" — близкие к оптимальному значения достигаются при m в диапазоне от 200 до 500. На практике получение подобного диапазона подходящих значений m зависит от характеристик используемых внешних запоминающих устройств и от длины записей в таблице.

В. И. Ландауэр (W. I. Landauer) [*IEEE Trans.* **EC-12** (1963), 863–871] предложил строить m-арные деревья следующим образом: размещать узлы на уровне $l + 1$, лишь если уровень l почти заполнен. Эта схема требует довольно сложной системы поворотов, так как для вставки одного нового элемента может потребоваться основательная перестройка дерева. Ландауэр исходил из предположения, что поиск приходится выполнять гораздо чаще вставок и удалений.

Если файл хранится на диске и является объектом сравнительно редких вставок и удалений, для его представления подходит трехуровневое дерево, где первый уровень разветвления определяет, какой цилиндр диска будет использоваться, второй уровень разветвления определяет дорожку на этом цилиндре, а третий уровень содержит собственно записи. Такой метод называется *индексно-последовательной* организацией файла [см. *JACM* **16** (1969), 569–571].

Р. Мюнц (R. Muntz) и Р. Узгалис (R. Uzgalis) [*Proc. Princeton Conf. on Inf. Sciences and Systems* **4** (1970), 345–349] предложили изменить алгоритм поиска по дереву со вставкой 6.2.2T таким образом, что все вставки порождают узлы, относящиеся к той же странице, что и их родительский узел (если только это возможно) Если страница заполнена, начинается новая страница (если таковая имеется). Если количество страниц не ограничено и если данные поступают в случайном порядке, можно показать, что среднее количество обращений к страницам примерно равно

$H_N/(H_m - 1)$, что лишь немногим больше числа обращений при использовании наилучшего m-арного дерева (см. упр. 8).

B-деревья (B-trees). Новый подход ко внешнему поиску с помощью сильноветвящихся деревьев был открыт в 1972 году Р. Байером (R. Bayer), Э. Мак-Крейтом (E. McCreight) [*Acta Informatica* **1** (1972), 173–189] и независимо от них в то же время М. Кауфманом (M. Kaufman) [не опубликовано]. Их идея, которая основана на изменчивости нового вида структур данных, называемых *B-деревьями*, делает возможным поиск и обновление больших файлов с гарантированной эффективностью в наихудшем случае с помощью сравнительно простых алгоритмов.

B-деревом m-го порядка называется дерево, удовлетворяющее следующим условиям.

i) Каждый узел имеет не более m потомков.

ii) Каждый узел, за исключением корневого узла и листьев, имеет не менее $m/2$ потомков.

iii) Корневой узел, если он не является листом, имеет минимум двух потомков.

iv) Все листья находятся на одном и том же уровне и не содержат информации.

v) Нелистовой узел с k потомками содержит $k - 1$ ключей.

(Как обычно, под листом подразумевается конечный, не имеющий потомков узел. Поскольку листья не несут информации, их можно рассматривать как внешние узлы, которых нет в дереве, так что указатели на листы равны Λ.)

На рис. 30 показано B-дерево порядка 7. Каждый узел (за исключением корня и листьев) имеет от $\lceil 7/2 \rceil$ до 7 потомков, т. е. содержит 3, 4, 5 или 6 ключей. Корневой узел может содержать от 1 до 6 ключей; в приведенном на рисунке случае их 2. Все листья расположены на уровне 3. Заметьте, что (a) ключи расположены в порядке возрастания слева направо с использованием естественного обобщения концепции симметричного порядка; (b) количество листьев на единицу превышает количество ключей.

B-деревья порядка 1 и 2, очевидно, не представляют интереса, поэтому будем рассматривать только случай, когда $m \geq 3$. 2-3-деревья, описанные в разделе 6.2.3, представляют собой B-деревья порядка 3.

(Байер и Мак-Крейт рассматривали только нечетные m; некоторые авторы, говоря о B-деревьях порядка m, подразумевают под ними то, что мы назвали бы деревом порядка $2m + 1$.)

Узел, содержащий j ключей и $j + 1$ указатель, может быть представлен как

$$\mathsf{P}\downarrow \quad \left(\mathsf{P}_0, K_1, \mathsf{P}_1, K_2, \mathsf{P}_2, \ldots, \mathsf{P}_{j-1}, K_j, \mathsf{P}_j\right), \tag{1}$$

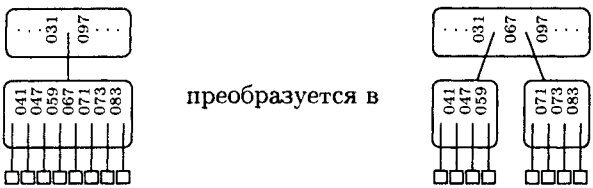
где $K_1 < K_2 < \cdots < K_j$, а P_i указывает на поддерево с ключами между K_i и K_{i+1}. Таким образом, поиск в B-дереве прямолинеен: после размещения узла (1) во внутренней памяти выполняется поиск данного аргумента среди ключей $K_1, K_2, \ldots, K_j$. (При больших j, вероятно, удобнее воспользоваться бинарным поиском; при малых j наилучшим способом поиска будет последовательный поиск.) Если поиск успешен, значит, искомое найдено; при неудачно завершенном поиске, когда ключ находится

между K_i и K_{i+1}, следует загрузить в оперативную память узел, на который указывает P_i, и продолжить процесс. Если аргумент поиска меньше K_1, используется указатель P_0; при аргументе, большем K_j, для продолжения поиска применяется указатель P_j. Если $\mathsf{P}_i = \Lambda$, значит, поиск завершился неудачно.

Приятная особенность B-деревьев заключается в простоте вставки. Рассмотрим, например, рис. 30. Каждый лист соответствует месту, где может быть выполнена новая вставка. Чтобы вставить новый ключ 337, просто заменим соответствующий узел

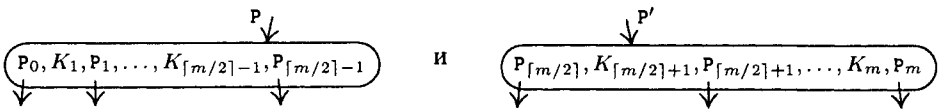
(2)

С другой стороны, при попытке вставить новый ключ 071 (программистам на C/C++: не примите случайно это число (071) за восьмеричную запись числа 57 :-). — *Прим. перев.*) можно обнаружить, что места для него нет, поскольку соответствующий узел на втором уровне заполнен до отказа. Однако эта неприятность легко преодолевается путем разбиения узла на две части с тремя ключами в каждой и передачи среднего ключа на первый уровень:



(3)

В целом, чтобы вставить новый элемент в B-дерево порядка m, в котором все листья расположены на уровне l, необходимо вставить новый ключ в подходящее место на уровне $l-1$. Если соответствующий узел содержит m ключей, т. е. имеет вид (1) с $j = m$, разбиваем его на два узла



(4)

и вставляем ключ $K_{\lceil m/2 \rceil}$ в родительский по отношению к данному узлу узел (другими словами, указатель P в родительском узле заменяется последовательностью P, $K_{\lceil m/2 \rceil}$, P′). Такая вставка может привести к аналогичному делению родительского узла (см. рис. 27, на котором показан случай для $m = 3$). При необходимости разделить корневой узел — сироту без родителя — просто создаем новый корневой узел, содержащий единственный ключ $K_{\lceil m/2 \rceil}$; дерево при этом прибавляет в росте и становится на единицу выше.

При такой процедуре вставки все свойства B-деревьев сохраняются; чтобы оценить всю красоту идеи, рекомендуем выполнить упр. 1. По существу, дерево растет "сверху вверх", вместо того чтобы делать это "снизу вниз", поскольку единственная причина, вызывающая рост дерева, — разделение корня.

Операция удаления из B-дерева немногим сложнее вставки (см. упр. 6).

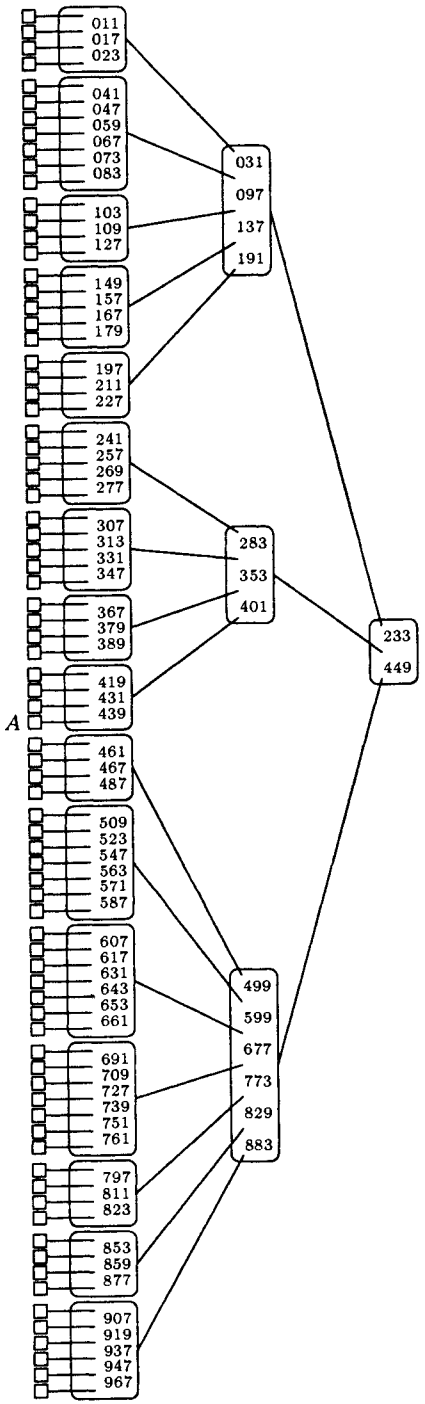


Рис. 30. B-дерево порядка 7 с листьями, расположенными на третьем уровне. Каждый узел содержит 3, 4, 5 или 6 ключей. Лист, предшествующий ключу 449, помечен символом A (см. (8)).

Верхняя граница времени работы. Посмотрим теперь, к скольким узлам потребуется обратиться в наихудшем случае при поиске в B-дереве порядка m. Предположим, что имеется N ключей и на уровне l имеется $N+1$ лист. Тогда количество узлов на уровнях $1, 2, 3, \ldots$ равно как минимум 2, $2\lceil m/2 \rceil$, $2\lceil m/2 \rceil^2$, $\ldots$; следовательно,

$$N + 1 \geq 2\lceil m/2 \rceil^{l-1}. \tag{5}$$

Другими словами,

$$l \leq 1 + \log_{\lceil m/2 \rceil}\left(\frac{N+1}{2}\right); \tag{6}$$

это означает, например, что если $N = 1\,999\,998$ и $m = 199$, то l не превышает 3. Поскольку в процессе поиска нам необходим доступ по крайней мере к l узлам, данная формула гарантирует, что время работы алгоритма весьма мало́.

При вставке нового узла может потребоваться разделить до l узлов, однако среднее количество таких разделений существенно меньше. Общее количество разделений равно общему количеству внутренних узлов минус l. При наличии в дереве p внутренних узлов имеется как минимум $1 + \left(\lceil m/2 \rceil - 1\right)(p-1)$ ключей. Следовательно,

$$p \leq 1 + \frac{N-1}{\lceil m/2 \rceil - 1}. \tag{7}$$

Таким образом, среднее количество разделений узлов при построении дерева, содержащего N ключей, меньше $1/\left(\lceil m/2 \rceil - 1\right)$ разделений на узел.

Усовершенствования и изменения. Существует несколько возможностей улучшить методы работы с B-деревьями при небольших изменениях их определения.

Прежде всего, заметим, что все указатели на уровне $l-1$ равны Λ, а все указатели на других уровнях не равны Λ. Зачастую это приводит к значительной потере пространства, так что можно "покорить пространство и время", убрав все Λ и использовав другое значение m для всех "нижних" узлов. Подобное применение двух различных m, однако, не ухудшает алгоритма вставки, так как обе половины разделенного узла остаются на том же уровне, что и оригинал. Можно даже определить обобщенное B-дерево порядков $m_1, m_2, m_3, \ldots$, потребовав, чтобы все некорневые узлы на уровне $l-1$ имели от $m_k/2$ до m_k потомков. Такое B-дерево имеет различные m на каждом уровне, однако алгоритм вставки при этом, в сущности, не меняется.

Развивая изложенную в предыдущем абзаце идею, можно использовать различные форматы узлов на различных уровнях дерева, а также хранить информацию в листьях. Иногда ключи занимают лишь малую часть записи в файле, и в таких случаях хранение записей целиком в близких к корню узлах может оказаться ошибкой: это может привести к слишком малым m и снизить тем самым эффективность сильного ветвления.

Обратимся вновь к рис. 30 и представим себе, что все записи файла хранятся в листьях и лишь несколько ключей продублировано в узлах ветвей. В таком случае крайний слева лист содержит все записи, для которых ключ ≤ 011; лист, помеченный символом A, содержит записи, удовлетворяющие соотношению

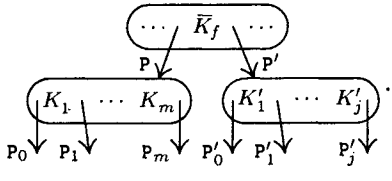
$$439 < K \leq 449 \tag{8}$$

и т. д. При этом листья растут и расщепляются, как и другие узлы, с тем отличием, что записи никогда не переходят из листьев на верхние уровни и, таким образом, листья всегда заполнены хотя бы наполовину. Если каждый лист связан со следующим в симметричном порядке листом, то можно эффективно и удобно проходить по файлу как последовательно, так и в случайном порядке. Такой вариант дерева называется B^+-деревом.

Некоторые вычисления, проведенные С. П. Гошем (S. P. Ghosh) и М. Э. Сенко (M. E. Senko) [*JACM* **16** (1969), 569–579], приводят к мысли о том, что неплохо иметь большие листья длиной, скажем, до 10 последовательных страниц. При помощи линейной интерполяции в известном для каждого листа диапазоне ключей можно предположить, в какой из десяти страниц должен находиться данный аргумент. Если такая догадка окажется неверной, будет потеряно время, однако эксперименты свидетельствуют о том, что эти потери могут оказаться меньшими, чем время, сэкономленное на уменьшении дерева.

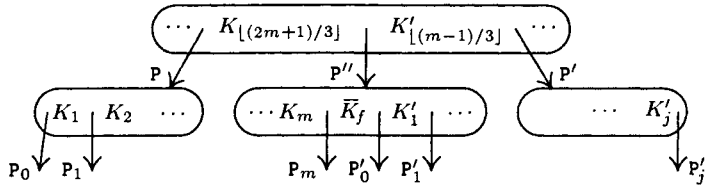
Т. Х. Мартин (T. H. Martin) [неопубликовано] указал, что идея, лежащая в основе B-деревьев, также может использоваться для ключей *переменной длины*. Нет необходимости требовать, чтобы количество потомков каждого узла находилось в пределах $[m/2\,..\,m]$; вместо этого можно просто сказать, что каждый узел должен быть заполнен данными как минимум наполовину. Механизм вставки и разделения продолжает неплохо работать, хотя точное количество ключей в узле зависит от их длин. Однако ключи не должны быть очень длинными, иначе это может все испортить (см. упр. 5).

Другой важной модификацией схемы B-дерева является идея *переливания*, предложенная Байером (Bayer) и Мак-Крейтом (McCreight). Она заключается в улучшении алгоритма вставки путем сопротивления соблазну частого разделения. Вместо этого предлагается использовать локальные повороты. Предположим, что имеется переполненный узел, содержащий m ключей и $m+1$ указатель. Вместо его разделения можно сначала обратить внимание на соседний узел справа, у которого, к примеру, есть j ключей и $j+1$ указатель. В родительском узле имеется ключ $\overline{K}_f$, который разделяет ключи двух соседей. Схематично это можно изобразить так:



(9)

При $j < m-1$ можно избежать разделения с помощью простой перегруппировки: оставляем $\lfloor (m+j)/2 \rfloor$ ключей в левом узле, заменяем в родительском узле $\overline{K}_f$ на $K_{\lfloor (m+j)/2 \rfloor + 1}$ и помещаем $\lceil (m+j)/2 \rceil$ оставшихся ключей (в том числе $\overline{K}_f$) и соответствующие указатели в правый узел. Таким образом, заполненный узел "переливается" в соседний. С другой стороны, если соседний узел также заполнен ($j = m-1$), можно разделить *оба* узла, сделав из них три узла, заполненных на две трети и содержащих соответственно $\lfloor (2m-2)/3 \rfloor$, $\lfloor (2m-1)/3 \rfloor$ и $\lfloor 2m/3 \rfloor$ ключей:



(10)

Если у исходного узла нет соседа справа, можно обратиться с предложением о переливании к соседу слева (если имеются оба соседа, то избегать разделения узла можно до тех пор, пока они *оба* не заполнятся). И, наконец, если исходный узел не имеет соседей, значит, это корневой узел. Можно изменить определение B-дерева, позволив корню содержать до $2\lfloor(2m-2)/3\rfloor$ ключей, чтобы при разделении корень давал два узла, содержащих по $\lfloor(2m-2)/3\rfloor$ ключей. (Не рассмотренный автором случай переливания к соседу, родительский узел которого является соседом родителя исходного узла, хотя и возможен теоретически, но слишком сложен, чтобы иметь преимущество перед разделением узлов. — *Прим. перев.*)

В результате применения всех описанных технологий выводится новая "порода" деревьев (назовем их B^*-деревьями порядка m), которые можно определить следующим образом.

i) Каждый узел, за исключением корневого, имеет не более m потомков.

ii) Каждый узел, за исключением корневого и листьев, имеет не менее $(2m-1)/3$ потомков.

iii) Корневой узел имеет не менее 2 и не более $2\lfloor(2m-2)/3\rfloor+1$ потомков.

iv) Все листья находятся на одном уровне.

v) Узлы, не являющиеся листьями и имеющие k потомков, содержат $k-1$ ключ.

Важным изменением является условие (ii), которое утверждает, что используется минимум две трети имеющегося в каждом узле пространства. Это изменение позволяет не только более эффективно использовать пространство, но и повысить скорость работы, поскольку при этом в формулах (6) и (7) $\lceil m/2\rceil$ заменяется на $\lceil(2m-1)/3\rceil$. Однако процесс вставки становится более медленным, так как узлам приходится уделять больше внимания при их заполнении (см. приближенный анализ в работе B. Zhang and M. Hsu, *Acta Informatica* **26** (1989), 421–438).

Иногда выгоднее позволить узлам быть заполненными менее чем наполовину, чем часто их изменять. Такая ситуация была проанализирована в работе T. Johnson and D. Shasha, *J. Comput. Syst. Sci.* **47** (1993), 45–76.

Возможно, читатель скептично настроен по отношению к B-деревьям, поскольку степень ветвления в корне может быть равна 2. Какой смысл тратить обращение к диску для разветвления всего лишь на 2 пути?! Однако простейшая схема буферизации, называемая *замещением дольше всего не используемой страницы*, позволяет преодолеть это неудобство. Можно выделить во внутренней памяти несколько буферов и избежать реального ввода с диска при наличии страницы в памяти. При использовании этой схемы алгоритмы поиска или вставки выполняют команды "виртуального чтения", которые транслируются в реальные команды ввода данных с диска только при отсутствии необходимой страницы в памяти. Последующая команда "освобождения" выполняется в том случае, когда информация в буфере модифицируется алгоритмом. Когда требуется действительное чтение, выбирается

буфер, который был освобожден ранее других, его содержимое, если оно было изменено с момента чтения, записывается на диск и затем в него считывается необходимая страница.

Поскольку число уровней дерева обычно мало́ по сравнению с количеством буферов, такая страничная схема обеспечивает постоянное наличие корневой страницы в памяти; если корень имеет 2–3 потомка, то же самое можно сказать и о страницах первого уровня. Заметим, что страницы, которые могут понадобиться для разделения при вставке, автоматически присутствуют в памяти в силу обращения к ним во время предыдущего поиска.

Эксперименты Э. Мак-Крейта (E. McCreight) показали, что такая политика работы вполне успешна. Например, им было найдено, что при 10 буферах и $m = 121$ процесс вставки 100 000 ключей в порядке возрастания требует только 22 действительных чтений и только 857 реальных команд записи. Таким образом, бо́льшая часть действий выполнялась во внутренней памяти. Далее, полученное дерево содержало всего 835 узлов, лишь на один узел больше, чем минимально возможное значение $\lceil 100000/(m-1) \rceil = 834$; следовательно, память была использована практически на 100%. В этом эксперименте применялась технология переливания, но с разделением на две части согласно (4), а не на три, как в (10) (см. упр. 3).

В другом эксперименте, также с десятью буферами, $m = 121$ и использованной технологией переливания, Э. Мак-Крейт вставил 5 000 ключей в первоначально пустое дерево в *случайном* порядке. При этом было получено двухуровневое дерево с 48 узлами (утилизация памяти составила 87%) после выполнения 2 762 реальных чтений и 2 739 реальных записей. Затем для 1 000 случайных поисков потребовалось 786 реальных чтений. В результате того же эксперимента *без* применения переливания было построено двухуровневое дерево с 62 узлами (утилизация памяти — 67%) после выполнения 2 743 реальных чтений и 2 800 реальных записей. Далее для 1 000 случайных поисков потребовалось 836 реальных чтений. Это показывает не только эффективность страничной схемы, но и преимущества, полученные от использования технологии переливания.

Эндрю Яо (Andrew Yao) доказал, что среднее количество узлов после случайных вставок без переливания для больших N и m равно

$$N/(m \ln 2) + O(N/m^2),$$

так что утилизация памяти будет составлять примерно $\ln 2 = 69.3\%$ [*Acta Informatica* **9** (1978), 159–170]. Более детальный анализ можно найти в работах B. Eisenbarth, N. Ziviani, G. H. Gonnet, K. Mehlhorn, and D. Wood, *Information and Control* **55** (1982), 125–174; R. A. Baeza-Yates, *Acta Informatica* **26** (1989), 439–471.

B-деревья со времен своего изобретения стали чрезвычайно популярны. Обратитесь, например, к статье Дугласа Комера (Douglas Comer) в *Computing Surveys* **11** (1979), 121–138, 412, в которой обсуждаются ранние разработки и описывается широко используемая система VSAM (Virtual Storage Access Method — метод доступа к виртуальной памяти), созданная в IBM Corporation. Одним из новшеств VSAM была репликация блоков дисковых цилиндров с целью минимизации времени обращения.

Две из наиболее интересных разработок основ стратегии B-деревьев получили одинаковые имена: "SB-деревья" и "SB-деревья". SB-дерево П. Ю. О'Нейла

(P. E. O'Neil) [*Acta Inf.* **29** (1992), 241–265] создано для минимизации времени выполнения дисковых операций путем расположения соседних записей в одном цилиндре; тем самым поддерживается высокая эффективность работы приложений, в которых требуется одновременный доступ к множеству последовательных записей (в этом случае "*SB*" выделяется курсивом и *S* означает "sequential" — "последовательный"). SB-дерево (см. P. Ferragina and R. Grossi, *STOC* **27** (1995), 693–702; *SODA* **7** (1996), 373–382) представляет собой элегантную комбинацию структуры *B*-дерева с деревьями "Патриция", о которых пойдет речь в разделе 6.3; в этом случае "SB" записывается обычным шрифтом и S означает "string" — "строка". Такие деревья используются во многих приложениях для крупномасштабной обработки текстов и обеспечивают эффективную сортировку строк переменной длины на диске [Arge, Ferragina, Grossi, and Vitter, *STOC* **29** (1997), 540–548].

УПРАЖНЕНИЯ

1. [*10*] Какое *B*-дерево порядка 7 получится после вставки ключа 613 в дерево, показанное на рис. 30 (технология переливания не используется)?

2. [*15*] Выполните упр. 1, используя технологию переливания с разделением на три части, как в (10).

▶ **3.** [*23*] Предположим, что ключи 1, 2, 3, ... в порядке возрастания вставляются в изначально пустое *B*-дерево порядка 101. Какой ключ приведет к появлению листьев на уровне 4:

a) без использования переливания;

b) при использовании переливания с разделением только на 2 части, как в (4);

c) при использовании B^*-дерева порядка 101 с переливанием и разделением на 3 части, как в (10)?

4. [*21*] (Байер (Bayer) и Мак-Крейт (McCreight).) Объясните, как выполнить вставки в обобщенное *B*-дерево так, чтобы все узлы, кроме корня и листьев, имели не менее $\frac{3}{4}m - \frac{1}{2}$ потомков.

▶ **5.** [*21*] Предположим, что для узлов отведено по 1 000 символов во внешней памяти. Если каждый указатель занимает 5 символов и если ключ имеет длину, кратную пяти и находящуюся в пределах от 5 до 50 символов, то каково минимальное количество символов, занятых в узле после его разделения в процессе вставки? (Рассмотрите только простую процедуру разделения, аналогичную описанной в тексте для *B*-дерева с ключом фиксированной длины без переливания; рассмотрите перемещение вверх ключа, который делает оставшиеся части наиболее близкими по размеру.)

6. [*23*] Разработайте алгоритм удаления из *B*-деревьев.

7. [*28*] Разработайте алгоритм конкатенации *B*-деревьев (см. раздел 6.2.3).

▶ **8.** [*HM37*] Рассмотрим обобщение вставки в дерево, предложенное Мюнцем (Muntz) и Узгалисом (Uzgalis), в котором каждая страница может содержать M ключей. Пусть в дерево вставлено N случайных элементов, так что оно имеет $N+1$ внешний узел. Обозначим через $b_{Nk}^{(j)}$ вероятность того, что при неудачном поиске потребуется k обращений к страницам и что он закончится в узле, родительский узел которого принадлежит странице, содержащей j ключей. Пусть $B_N^{(j)}(z) = \sum b_{Nk}^{(j)} z^k$ — соответствующая производящая

функция. Докажите, что $B_1^{(j)}(z) = \delta_{j1} z$ и

$$B_N^{(j)}(z) = \frac{N-j-1}{N+1} B_{N-1}^{(j)}(z) + \frac{j+1}{N+1} B_{N-1}^{(j-1)}(z) \qquad \text{для } 1 < j < M;$$

$$B_N^{(1)}(z) = \frac{N-2}{N+1} B_{N-1}^{(1)}(z) + \frac{2z}{N+1} B_{N-1}^{(M)}(z);$$

$$B_N^{(M)}(z) = \frac{N-1}{N+1} B_{N-1}^{(M)}(z) + \frac{M+1}{N+1} B_{N-1}^{(M-1)}(z).$$

Найдите асимптотическое поведение $C'_N = \sum_{j=1}^{M} B_N^{(j)\prime}(1)$ — среднего числа страниц, к которым осуществляется обращение при неудачном поиске. [*Указание.* Выразите рекуррентное соотношение с помощью матрицы

$$W(z) = \begin{pmatrix} -3 & 0 & \dots & 0 & 2z \\ 3 & -4 & \dots & 0 & 0 \\ 0 & 4 & \dots & 0 & 0 \\ \vdots & \vdots & & \vdots & \vdots \\ 0 & 0 & \dots & -M-1 & 0 \\ 0 & 0 & \dots & M+1 & -2 \end{pmatrix}$$

и сопоставьте C'_N с многочленом N-й степени в $W(1)$].

9. [*22*] Может ли идея B-деревьев использоваться для получения элементов линейного списка по позиции, а не по значению ключа (см. алгоритм 6.2.3B)?

▶ **10.** [*35*] Обдумайте, каким образом большой файл, организованный в виде B-дерева, можно использовать для доступа и изменения со стороны большого числа одновременно работающих пользователей, чтобы пользователи различных страниц практически не мешали друг другу.

Мало известно, даже при использовании иных эквивалентных алгоритмов, об оптимизации выделения памяти, минимизации количества необходимых операций и т. п. Эта область исследований должна черпать ресурсы для дальнейшего прогресса как из чистой, так и из прикладной математики.

— ЭНТОНИ Г. ЁТТИНГЕР (ANTHONY G. OETTINGER) (1961)

6.3. ЦИФРОВОЙ ПОИСК

ВМЕСТО методов поиска, основанных на сравнении ключей, можно воспользоваться представлением ключей в виде последовательности цифр или букв. Рассмотрим, например, "побуквенные метки" в больших словарях (или записных адресных книжках), которые позволяют мгновенно найти страницы со словами, начинающимися с определенной буквы*.

Развивая идею побуквенных меток до ее логического завершения, можно получить схему поиска, основанную на повторяемом индексировании, которое проиллюстрировано в табл. 1. Предположим, что необходимо проверить, является ли данный аргумент поиска одним из 31 наиболее употребительного английского слова (см. рис. 12 и 13 в разделе 6.2.2). Данные представлены в табл. 1 в виде *структуры луча***. Луч, по сути, — это M-арное дерево, узлы которого представляют M-местные векторы с компонентами, соответствующими цифрам или буквам. Каждый узел уровня l является набором всех ключей, начинающихся с определенной последовательности l символов, которая называется его *префиксом* (*prefix*); узел определяет разветвление на M путей в зависимости от $l+1$ символа.

Например, луч из табл. 1 содержит 12 узлов; узел (1) представляет собой корень, в котором мы ищем первую букву. Если первой буквой является, скажем, N, из таблицы следует, что либо это слово NOT, либо такого слова вообще нет в таблице. В то же время, если первая буква — W, то узел (1) отправит нас к узлу (9) для поиска второй буквы тем же способом. Узел (9) указывает, что вторая буква должна быть A, H или I. Префикс узла (10) — HA. Пустые записи в таблице означают отсутствие связей.

Узлы-векторы в табл. 1 расположены в соответствии с кодами символов MIX. Это означает, что "лучевой поиск" будет весьма быстрым, поскольку следует просто выбирать слова из массивов с использованием символов наших ключей в качестве индексов. Технологии быстрого многопутевого принятия решения по индексу называются просмотром таблицы (table look-at) в отличие от поиска по таблице (table look-up) [см. P. M. Sherman, *CACM* **4** (1961), 172–173, 175].

Алгоритм T (*"Лучевой поиск"* (*Trie search*)). Дана таблица записей в форме M-арного луча. Алгоритм осуществляет поиск по заданному аргументу K. Узлы луча представляют собой векторы, индексы которых изменяются от 0 до $M-1$; каждый компонент такого вектора представляет собой ключ или ссылку (возможно, пустую).

T1. [Инициализация.] Установить ссылочную переменную P таким образом, чтобы она указывала на корень луча.

T2. [Ветвление.] Установить k равным следующему символу входного аргумента ключа K слева направо. (Если аргумент полностью просканирован, установить

---

* Наличие у страницы книги трех свободных сторон позволило однажды переводчику этого раздела оснастить свой словарь метками поиска до третьей буквы слова включительно. — *Прим. перев.*

** В оригинале используется термин *trie* (произносится как английское слово "try"), предложенный Э. Фредкиным (E. Fredkin) [*CACM* **3** (1960), 490–500]. Этот термин представляет собой часть слова "re*trie*val". В русском переводе будет использоваться часть слова "по*луч*ение" (информации). И хотя при этом теряется определенная игра слов, основанная на схожести слов *trie* и *tree*, приобретается некоторый смысл, заложенный в слове *луч*: быстрый четко определенный по направлению движения поиск. — *Прим. перев.*

k равным "пустому" символу или символу "конец слова". Символ должен быть представлен в виде числа в диапазоне $0 \le k < M$). Обозначим через X k-й элемент в NODE(P). Если X представляет собой ссылку, перейти к шагу T3, если X — ключ, перейти к шагу T4.

T3. [Продвижение.] Если $X \neq \Lambda$, установить P $\leftarrow X$ и вернуться к шагу T2; в противном случае алгоритм завершается неудачей.

T4. [Сравнение.] Если $X = K$, алгоритм завершается успешно; в противном случае алгоритм завершается неудачей. ▌

Таблица 1

ЛУЧ ДЛЯ 31 НАИБОЛЕЕ УПОТРЕБИТЕЛЬНОГО АНГЛИЙСКОГО СЛОВА

	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)	(11)	(12)
␣		A				I					HE	
A	(2)				(10)				WAS			THAT
B	(3)											
C												
D										HAD		
E			BE		(11)							THE
F	(4)						OF					
G												
H	(5)							(12)	WHICH			
I	(6)				HIS				WITH			THIS
Θ												
J												
K												
L												
M												
N	NOT	AND				IN	ON					
O	(7)			FOR				TO				
P												
Q												
R		ARE		FROM			OR				HER	
Φ												
Π												
S		AS				IS						
T	(8)	AT				IT						
U			BUT									
V										HAVE		
W	(9)											
X												
Y	YOU		BY									
Z												

Заметим, что при неудачном поиске будет найден элемент, *более всего совпадающий* с искомым, что может оказаться полезным в некоторых приложениях.

Для сравнения скорости работы данного алгоритма со скоростями других алгоритмов из этой главы напишем коротенькую MIX-программу, в которой предполагается, что символы представляют собой байты, а максимальная длина ключа — 5 байт.

Программа T (*"Лучевой поиск"* (*Trie search*)). В этой программе предполагается, что все ключи занимают одно слово машины MIX; в случае, если в ключе содержится

менее 5 символов, он дополняется пробелами справа. Поскольку мы используем коды символов MIX, каждый байт аргумента поиска содержит число, меньшее 30. Ссылки представлены в виде отрицательных чисел в поле 0:2 узла слова. rI1 ≡ P, rX ≡ непросканированная часть K.

```
01  START   LDX  K          1   T1. Инициализация.
02          ENT1 ROOT       1   P ← указатель на корень луча.
03  2H      SLAX 1          C   T2. Ветвление.
04          STA  *+1(2:2)   C   Получение следующего символа, т. е. k.
05          ENT2 0,1        C   Q ← P + k.
06          LD1N 0,2(0:2)   C   P = LINK(Q).
07          J1P  2B         C   T3. Продвижение. Переход к T2, если P≠Λ.
08          LDA  0,2        1   T4. Сравнение. rA ← KEY(Q).
09          CMPA K          1
10          JE   SUCCESS    1   Успешное завершение при rA = K.
11  FAILURE EQU  *              Выход при отсутствии в луче. ▌
```

Время работы программы составляет $8C + 8$ единиц, где C — количество проверяемых символов. Поскольку $C \leq 5$, время поиска никогда не превышает 48 единиц времени.

Если сравнить эффективность этой программы (при поиске на луче из табл. 1) и программы 6.2.2T (с использованием *оптимального* бинарного дерева поиска (см. рис. 13)), можно сделать следующие выводы.

1. Луч занимает гораздо больше памяти; мы использовали 360 слов для представления 31 ключа, в то время как бинарное дерево поиска использует только 62 слова памяти (однако в упр. 4 будет показано, что можно ухитриться втиснуть луч из табл. 1 в 49 слов).
2. Успешный поиск требует 26 единиц времени в обеих программах. Неудачный поиск выполняется быстрее в случае луча и медленнее — в бинарном дереве поиска. Для наших конкретных данных неудачный поиск будет осуществляться гораздо чаще, чем успешный, а потому для них "лучевой поиск" предпочтительнее с точки зрения скорости работы.
3. В случае применения луча для указателя KWIC (см. рис. 15) метод теряет свою привлекательность в силу природы используемых данных. Например, для различения слов COMPUTATION и COMPUTATIONS луч потребует 12 итераций. Поэтому было бы более разумно строить луч таким образом, чтобы сканирование слов происходило справа налево.

Абстрактная концепция луча для представления семейства строк была предложена Акселем Тью (Axel Thue) в статье о строках, которые не содержат смежных повторяющихся подстрок [*Skrifter udgivne af Videnskabs-Selskabet i Christiania*, Mathematisk-Naturvidenskabelig Klasse (1912), No. 1; перепечатана в книге Тью *Selected Mathematical Papers* (Oslo: Universitetsforlaget, 1977), 413–477].

"Лучевая память" для компьютерного поиска была впервые рекомендована Рене де ла Брианде (René de la Briandais) [*Proc. Western Joint Computer Conf.* **15** (1959), 295–298]. Он указал, что можно сохранить память (за счет времени работы) при использовании связанного списка для каждого узла-вектора, поскольку большинство элементов вектора обычно пусто. На самом деле эта идея приводит к замещению луча из табл. 1 лесом, показанным на рис. 31. Поиск в таком лесу осуществляется

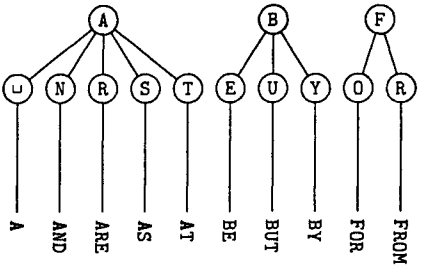
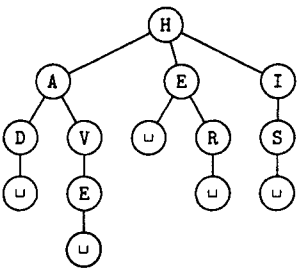


Рис. 31. Луч из табл. 1, преобразованный в "лес".

путем нахождения корня, соответствующего первому символу, затем дочернего узла этого корня, соответствующего второму символу, и т. д.

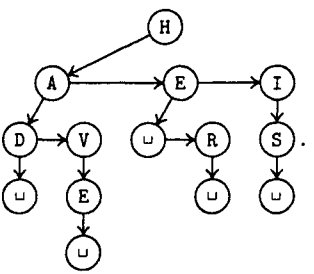
В своей статье де ла Брианде в действительности не прерывал ветвление дерева, как показано в табл. 1 или на рис. 31; вместо этого он продолжал представление каждого ключа, символ за символом, до достижения конца слова. Таким образом, де ла Брианде использовал бы



(1)

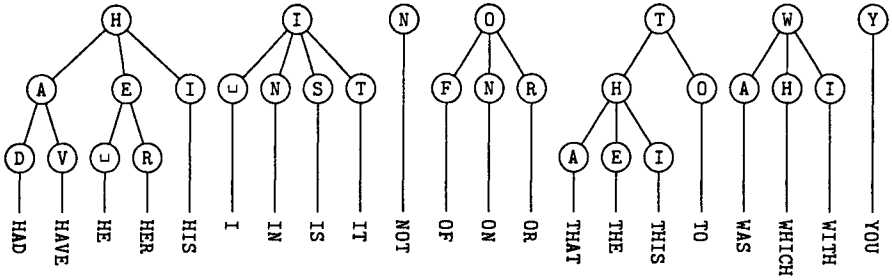
вместо дерева H, показанного на рис. 31. Для такого представления требуется больше памяти, но при этом существенно упрощается работа с ключами переменной длины. При использовании двух полей ссылок на каждый символ легко осуществляются динамическая вставка и удаление.

Если использовать обычный путь представления деревьев как бинарных деревьев, (1) становится бинарным деревом



(2)

(В представлении полного леса на рис. 31 следовало бы добавить указатель, ведущий вправо от H к соседнему корню I.) Поиск в этом бинарном дереве выполняется посредством сравнения аргумента с символом в дереве и следования RLINK до поиска соответствия; после этого мы находим LLINK и точно так же работаем с очередным символом аргумента.



Поиск в таком бинарном дереве в большей или меньшей степени сводится к поиску путем сравнений, но ветвление осуществляется по признаку "равно-не равно", а не "меньше-больше". Элементарная теория из раздела 6.2.1 гласит, что необходимо выполнить в среднем хотя бы $\lg N$ сравнений для того, чтобы различить N ключей. Среднее количество сравнений, сделанных при поиске в дереве, которое подобно изображенному на рис. 31, должно быть не меньше количества сравнений, выполняемых при бинарном поиске с использованием описанных в разделе 6.2 технологий.

С другой стороны, луч из табл. 1 способен выполнять M-путевое ветвление за один раз; мы увидим, что среднее время поиска для больших N включает всего около $\log_M N = \lg N / \lg M$ итераций при случайных входных данных. Мы также увидим, что "чистая" схема луча (подобная алгоритму T) требует, в целом, примерно $N/\ln M$ узлов для различения N случайных ключей; следовательно, общее количество необходимой памяти пропорционально $MN/\ln M$.

Из этих рассуждений становится ясно, что идея луча хороша только для нескольких первых уровней дерева. Повысить производительность можно за счет комбинирования двух стратегий: луча для нескольких первых символов и переключения на другую стратегию — для оставшихся. Например, Э. Г. Сассенгат (мл.) (E. H. Sussenguth, Jr.) [*CACM* **6** (1963), 272–279] предложил использовать посимвольную схему до достижения части дерева, в которой возможны, скажем, не более шести ключей, а затем проходить последовательно по этому списку. Далее мы увидим, что такая смешанная стратегия позволяет уменьшить количество узлов луча примерно в шесть раз без существенного изменения времени работы.

Т. Н. Турба (T. N. Turba) [*CACM* **25** (1982), 522–526] указал, что иногда при поиске с ключами переменной длины удобно иметь по одному поисковому дереву или лучу для каждой длины ключа.

Бинарный цифровой поиск. Допустим, что $M = 2$, и аргумент поиска сканируется по одному биту. Для этого случая разработаны два специальных интересных метода.

Первый метод, именуемый цифровым *поиском по дереву* (*digital tree search*), разработан Э. Г. Коффманом (E. G. Coffman) и Дж. Ивом (J. Eve) [*CACM* **13** (1970), 427–432, 436]. Его суть заключается в хранении полных ключей в узлах так же, как в алгоритмах поиска по дереву из раздела 6.2.2, но с использованием битов аргумента вместо результатов сравнения для выбора левой или правой ветви. На рис. 32 изображено бинарное дерево, построенное по этому методу путем вставки 31 наиболее употребительного английского слова в порядке уменьшения частот появления слов. Чтобы получить данные для иллюстрации метода, слова были

представлены в кодах символов MIX и конвертированы в двоичные числа с пятью битами на байт. Так, слово WHICH представляется битовой последовательностью 11010 01000 01001 00011 01000.

Для поиска слова WHICH на рис. 32 сравним его со словом THE в корне дерева. Так как они не совпадают и первый бит слова WHICH равен 1, мы двигаемся вправо и сравниваем его со словом OF. Слова опять не совпадают, и следующий бит слова WHICH равен 1, поэтому мы снова перемещаемся вправо и сравниваем наше слово WHICH со словом WITH, и т. д. Алфавитный порядок ключей в цифровом поиске по дереву больше не соответствует симметричному порядку узлов.

Интересно обратить внимание на разницу между деревьями, представленными на рис. 32 и 12 (из раздела 6.2.2), так как последнее дерево было построено тем же способом, но с использованием сравнения ключей вместо битов. Если принять во внимание данные частóты появления слов, цифровой поиск по дереву на рис. 32 потребует в среднем 3.42 сравнений при успешном завершении поиска. Это несколько лучше, чем в случае дерева, показанного на рис. 12, для которого требуется 4.04 сравнения (хотя, конечно же, время самого сравнения различно для этих двух ситуаций).

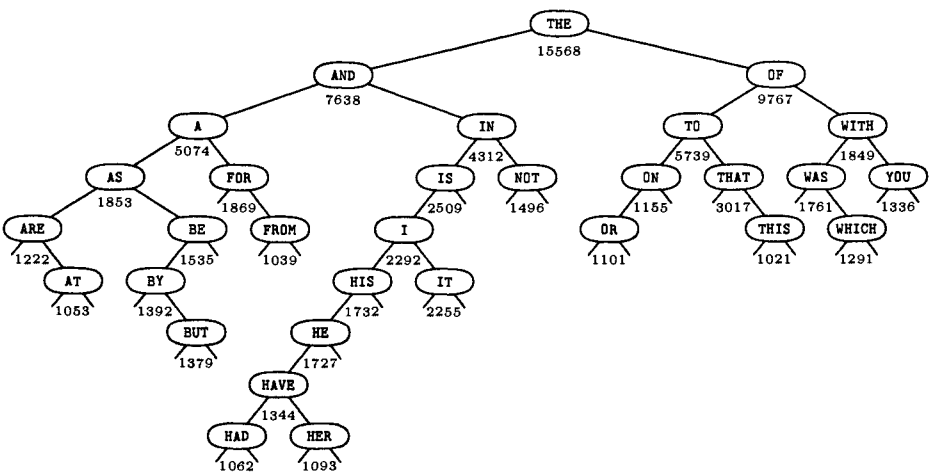


Рис. 32. Дерево цифрового поиска для 31 наиболее употребительного английского слова, вставленного в порядке уменьшения частот.

Алгоритм D (*Цифровой поиск со вставкой по дереву*). Дана таблица записей, представляющих бинарное дерево, которое описано выше. Алгоритм предназначен для поиска некоторого аргумента K. Если K отсутствует в таблице, новый узел, содержащий K, вставляется в дерево в соответствующем месте.

Алгоритм предполагает, что дерево не пусто и что его узлы имеют поля KEY, LLINK и RLINK, такие же, как в алгоритме 6.2.2T. В действительности, как вы можете убедиться сами, алгоритмы практически идентичны.

D1. [Инициализация.] Установить P $\leftarrow$ ROOT и $K' \leftarrow K$.

D2. [Сравнение.] Если K = KEY(P), поиск успешно завершается. В противном случае установить b равным лидирующему биту K' и сдвинуть K' влево на

один бит (тем самым удалив этот бит и вставив 0 справа). Если $b = 0$, перейти к шагу D3; в противном случае перейти к шагу D4.

D3. [Перемещение влево.] Если LLINK(P) $\neq \Lambda$, установить P $\leftarrow$ LLINK(P) и перейти к шагу D2. В противном случае перейти к шагу D5.

D4. [Перемещение вправо.] Если RLINK(P) $\neq \Lambda$, установить P $\leftarrow$ RLINK(P) и перейти к шагу D2.

D5. [Вставка в дерево.] Установить Q $\Leftarrow$ AVAIL, KEY(Q) $\leftarrow K$, LLINK(Q) $\leftarrow$ RLINK(Q) $\leftarrow \Lambda$. Если $b = 0$, присвоить LLINK(P) $\leftarrow$ Q; в противном случае присвоить RLINK(P) $\leftarrow$ Q. ▮

Хотя алгоритм поиска по дереву 6.2.2T по сути своей бинарный, не сложно увидеть, что он может быть распространен на M-арный цифровой поиск для любого $M \geq 2$ (см. упр. 13).

Дональд Р. Моррисон (Donald R. Morrison) [*JACM* **15** (1968), 514–534] открыл весьма привлекательный способ построения N-узловых деревьев поиска, основанных на бинарном представлении ключей *без* необходимости их хранения в узлах. Этот метод, названный "Патриция" (Patricia — **P**ractical **A**lgorithm **T**o **R**etrieve **I**nformation **C**oded **I**n **A**lphanumeric), очень хорошо подходит для работы с большими ключами переменной длины, например с заголовками или фразами, хранящимися в файле большого объема. Похожий алгоритм был одновременно опубликован в Германии (G. Gwehenberger, *Elektronische Rechenanlagen* **10** (1968), 223–226).

Основная суть метода "Патриция" состоит в построении бинарного дерева без однопутевых ветвей посредством включения в каждый узел количества битов, которые можно пропустить, прежде чем приступить к следующему тесту. Существует несколько способов реализации этой идеи; возможно, простейший из них представлен на рис. 33. Имеется массив битов TEXT (обычно довольно длинный), который может храниться во внешнем файле с произвольным доступом, поскольку при каждом поиске обращение к TEXT осуществляется только один раз. Каждый ключ, который должен храниться в нашей таблице, определяется местом его начала в тексте; можно считать, что он идет от места своего начала и до конца текста. (Метод "Патриция" не ищет точного соответствия между ключом и аргументом; вместо этого определяется, существует ли ключ, *начинающийся* с аргумента).

В ситуации, показанной на рис. 33, представлено семь ключей, которые начинаются с каждого входящего в текст слова, а именно — с "THIS IS THE HOUSE THAT JACK BUILT?", "IS THE HOUSE THAT JACK BUILT?", ..., "BUILT?". Имеется одно важное ограничение: *ни один ключ не может быть началом другого.* Оно выполняется, если текст завершается специальным символом конца текста (в нашем случае это "?"), который не встречается нигде в тексте. То же ограничение неявно применяется и в схеме луча алгоритма T, в которой признаком конца слова служит "␣". Дерево, используемое методом "Патриция" для поиска, должно целиком размещаться в оперативной памяти с произвольным доступом (или должно быть организовано по страничной схеме, описанной в разделе 6.2.4). Оно состоит из заголовка и $N - 1$ узла; узлы имеют несколько полей.

KEY, указатель на текст. Это поле должно иметь длину как минимум $\lg C$ бит, если текст содержит C символов. На рис. 33 слова, показанные внутри

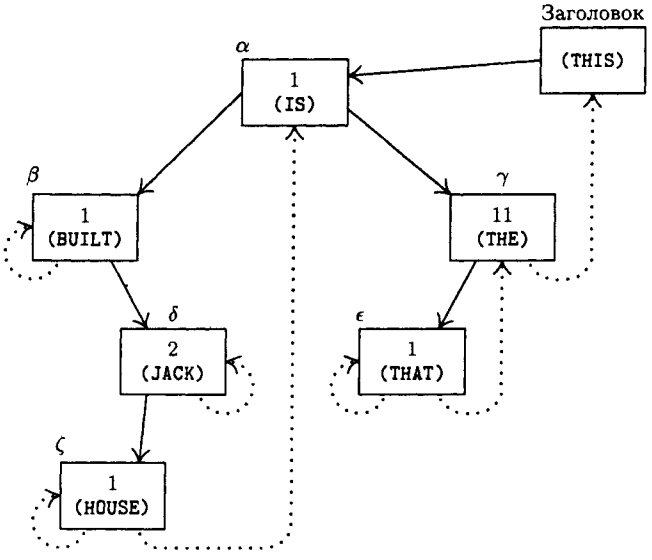


Рис. 33. Пример текста и дерева метода "Патриция".

каждого узла, на самом деле представлены указателями на текст, например вместо (JACK) узел содержит число 24, которое указывает начальное положение JACK BUILT? в текстовой строке.

LLINK и RLINK, ссылки внутри дерева. Длина этих полей должна составлять не менее $\lg N$ бит.

LTAG и RTAG, однобитовые поля, которые указывают, являются ли LLINK и RLINK ссылками на дочерние или родительские узлы данного узла соответственно. Пунктирные линии на рис. 33 соответствуют указателям, биты TAG которых равны 1.

SKIP, число, которое указывает, сколько битов должно быть пропущено при поиске (как объяснялось ранее). Это поле должно быть достаточно велико для хранения наибольшего числа k, для которого в двух различных ключах найдутся совпадающие подцепочки из k бит. Обычно на практике можно считать, что k не слишком велико, и сообщать об ошибке при превышении размеров поля SKIP. На рис. 33 поля SKIP показаны как числа внутри узлов.

Заголовок содержит только поля KEY, LLINK и LTAG.

Поиск в дереве метода "Патриция" выполняется следующим образом. Предположим, необходимо найти слово THE (его битовое представление — 10111 01000 00101). Начинаем просмотр с поля SKIP в корневом узле α, которое указывает, что следует проверить первый бит аргумента. Этот бит равен 1, и потому мы должны двигаться вправо. Поле SKIP следующего узла, γ, указывает, что теперь нужно обратить внимание на $1 + 11 = 12$-й бит аргумента. Он равен 0, поэтому мы движемся влево. Поле SKIP следующего узла, ϵ, заставляет нас взглянуть на $(12 + 1)$-й бит, равный 1.

Находим, что RTAG = 1, так что следует вернуться к узлу γ, который отсылает нас к массиву TEXT. Тот же путь поиска будет получен для любого аргумента, битовая маска которого равна 1xxxx xxxxx x01..., и необходимо проверить, не соответствует ли аргумент уникальному ключу, начинающемуся с этой маски, а именно — с THE.

Предположим, с другой стороны, что нужно найти некоторый ключ, начинающийся с TH (или все такие ключи). Процесс поиска начинается так же, как описывалось выше, но в конечном счете мы попытаемся обратиться к несуществующему двенадцатому биту десятибитового аргумента. Теперь необходимо сравнить аргумент с фрагментом массива TEXT, определяемым в текущем узле (в нашем случае — узле γ). Если совпадения не произошло, значит, аргумент не начинается ни с одного ключа; если же совпадение имело место, то аргумент служит началом любого ключа, на который указывают пунктирные линии, выходящие из узла γ и его потомков (а именно — THIS, THAT, THE).

Более точно процесс можно описать следующим образом.

Алгоритм P (*"Патриция"*). Дан массив TEXT и дерево с описанными выше полями KEY, LLINK, RLINK, LTAG, RTAG и SKIP. Алгоритм определяет, имеется ли в массиве TEXT ключ, начинающийся с некоторого аргумента K. (Если имеется $r \geq 1$ таких ключей, за $O(r)$ шагов можно последовательно установить расположение каждого из них; см. упр. 14.) Предполагается, что имеется, по меньшей мере, один ключ.

P1. [Инициализация.] Установить P $\leftarrow$ HEAD и $j \leftarrow 0$. (Переменная P представляет собой указатель, который будет перемещаться вниз по дереву, а j — счетчик, определяющий позиции битов аргумента.) Установить n равным количеству битов в аргументе K.

P2. [Движение влево.] Присвоить Q $\leftarrow$ P и P $\leftarrow$ LLINK(Q). Если LTAG(Q) = 1, перейти к шагу P6.

P3. [Пропуск битов.] (В этот момент известно, что если первые j бит K соответствуют некоторому ключу, то они соответствуют ключу, начинающемуся в KEY(P)). Установить $j \leftarrow j +$ SKIP(P). Если $j > n$, перейти к шагу P6.

P4. [Проверка бита.] (В этот момент известно, что если первые $j - 1$ бит аргумента соответствуют некоторому ключу, то они соответствуют ключу, начинающемуся в KEY(P)). Если j-й бит аргумента равен 0, перейти к шагу P2; в противном случае — к шагу P5.

P5. [Перемещение вправо.] Присвоить Q $\leftarrow$ P и P $\leftarrow$ RLINK(Q). Если RTAG(Q) = 0, перейти к шагу P3.

P6. [Сравнение.] (В этот момент известно, что если аргумент соответствует некоторому ключу, то он соответствует ключу, начинающемуся в KEY(P)). Сравнить K с ключом, начинающимся в позиции KEY(P) в массиве TEXT. Если они эквивалентны (до n-го бита (длины K)), то алгоритм успешно завершается; в случае неравенства он завершается неудачей. ▌

В упр. 15 показано, каким образом может быть построено дерево метода "Патриция". Можно также вносить дополнения к тексту и вставлять новые ключи, если новый текстовый материал всегда заканчивается уникальным разделителем (например, символом конца текста с последующим серийным номером).

Алгоритм "Патриция" несколько причудлив, и требуется внимание для выявления всех его достоинств.

Анализ алгоритмов. Завершается этот раздел математическим анализом лучей, деревьев цифрового поиска и метода "Патриция". Важнейшие выводы будут приведены в самом конце раздела.

Сначала рассмотрим бинарные лучи, т. е. лучи с $M = 2$. На рис. 34 показан бинарный луч, который был получен при рассмотрении шестнадцати ключей из примеров сортировки (глава 5) в качестве 10-битовых двоичных чисел. (Ключи показаны в *восьмеричной записи*, например *1144* представляет собой десятибитовое число $612 = (1001100100)_2$.) Как и в алгоритме Т, для хранения информации о ведущих битах ключей (до тех пор, пока ключ не идентифицируется однозначно) используется луч, в котором ключ записывается полностью.

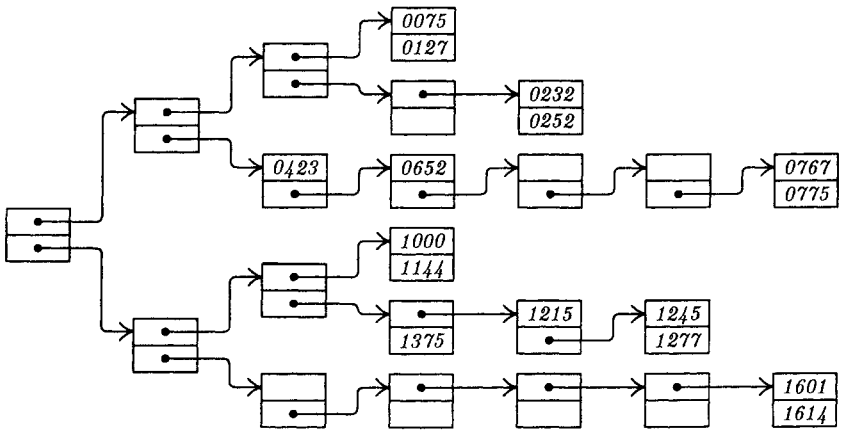


Рис. 34. Пример случайного бинарного луча.

Если сравнить рис. 34 с табл. 5.2.2–3, обнаружится удивительная взаимосвязь между "лучевой" памятью и обменной поразрядной сортировкой. (Возможно, эта связь очевидна.) 22 узла на рис. 34 точно совпадают с 22 стадиями из табл. 5.2.2–3 с соответствием p-го узла в предварительном порядке p-й стадии. Количество проверяемых на стадиях разбиения битов равно числу ключей в соответствующих узлах и их подлучах; значит, можно сформулировать следующий результат.

Теорема Т. *Если N различных двоичных чисел помещены в описанный выше бинарный луч, то* (i) *количество узлов луча равно количеству стадий разбиения при обменной поразрядной сортировке и* (ii) *среднее количество проверок битов, требующееся для выборки ключа с помощью алгоритма Т, равно $1/N$ от количества проверок битов при обменной поразрядной сортировке.* ▌

Благодаря этой теореме можно использовать весь математический аппарат, разработанный для поразрядной сортировки в разделе 5.2.2. Например, если предположить, что ключами являются случайные равномерно распределенные между 0 и 1 числа, заданные с бесконечной точностью, то количество проверок битов, необходимое для выборки, равно $\lg N + \gamma/\ln 2 + 1/2 + \delta(N) + O(N^{-1})$, а число узлов

луча — $N/\ln 2+N\bar{\delta}(N)+O(1)$. Здесь $\delta(N)$ и $\bar{\delta}(N)$ — сложные функции, которыми можно пренебречь, поскольку их абсолютные значения никогда не превышают 10^{-6} (см. упр. 5.2.2–38 и 5.2.2–48).

Конечно же, перед нами стоит более трудная задача: обобщить результаты, полученные для бинарных лучей, на случай M-арных лучей. Мы опишем лишь стартовую точку исследований, помещая детальные инструкции в упражнения.

Пусть A_N — среднее число внутренних узлов в случайном M-арном луче поиска, который содержит N ключей. Тогда $A_0 = A_1 = 0$ и для $N \geq 2$ имеем

$$A_N = 1 + \sum_{k_1+\cdots+k_M=N} \left(\frac{N!}{k_1!\ldots k_M!}M^{-N}\right)(A_{k_1}+\cdots+A_{k_M}), \tag{3}$$

поскольку $N!\,M^{-N}/k_1!\ldots k_M!$ представляет собой вероятность того, что k_1 ключей находятся в первом подлуче, ..., k_M — в M-м. Это соотношение может быть переписано как

$$\begin{aligned} A_N &= 1 + M^{1-N} \sum_{k_1+\cdots+k_M=N} \left(\frac{N!}{k_1!\ldots k_M!}\right) A_{k_1} \\ &= 1 + M^{1-N} \sum_k \binom{N}{k} (M-1)^{N-k} A_k \qquad \text{при } N \geq 2 \end{aligned} \tag{4}$$

с использованием симметрии и суммирования по $k_2, \ldots, k_M$. Аналогично, если через C_N обозначить общее среднее количество проверок битов, необходимое для поиска всех N ключей в луче, то $C_0 = C_1 = 0$ и

$$C_N = N + M^{1-N} \sum_k \binom{N}{k} (M-1)^{N-k} C_k \qquad \text{при } N \geq 2. \tag{5}$$

В упр. 17 показано, как работать с такого рода рекуррентными соотношениями, а в упр. 18–25 приводится соответствующая теория случайных лучей [с другой точки зрения анализ A_N был впервые проведен в работе L. R. Johnson, M. H. McAndrew, *IBM J. Res. and Devel.* **8** (1964), 189–193, в связи с эквивалентным аппаратно-ориентированным алгоритмом сортировки].

Если теперь перейти к изучению деревьев цифрового поиска, то обнаружится, что формулы похожи, однако не настолько, чтобы можно было легко определить асимптотическое поведение. Например, если через $\bar{C}_N$ обозначить среднее суммарное количество проверок битов при поиске всех N ключей в M-арном дереве цифрового поиска, нетрудно вывести, как и ранее, что $\bar{C}_0 = \bar{C}_1 = 0$ и

$$\bar{C}_{N+1} = N + M^{1-N} \sum_k \binom{N}{k} (M-1)^{N-k} \bar{C}_k \qquad \text{при } N \geq 0. \tag{6}$$

Это выражение практически идентично выражению (5), однако появления $N+1$ вместо N в левой части уравнения достаточно для того, чтобы коренным образом изменить характер рекуррентности, а потому использовавшиеся при изучении (5) методы становятся неприемлемыми.

Рассмотрим сначала бинарный цифровой поиск. На рис. 35 показано дерево цифрового поиска, соответствующее шестнадцати ключам из рис. 34, если они

вставляются в порядке, который использовался в примерах главы 5. Если будет необходимо определить среднее количество проверок битов при случайном успешном поиске, окажется, что это просто длина внутреннего пути дерева, деленная на N, так как для поиска узла на уровне l потребуется l проверок. Заметим, однако, что среднее число проверок битов при случайном *неудачном* поиске *не так* просто связано с длиной внешнего пути дерева, поскольку неудачный поиск с большей вероятностью оканчивается во внешнем узле недалеко от корня. Так, вероятность достижения левой ветви узла *0075* на рис. 35 равна $\frac{1}{8}$ (при рассмотрении ключей с бесконечной точностью), а вероятность попадания в левую ветвь узла *0232* составляет лишь $\frac{1}{32}$. Из-за этого деревья цифрового поиска при равномерном распределении ключей в целом лучше сбалансированы, чем бинарные деревья поиска из алгоритма 6.2.2Т.

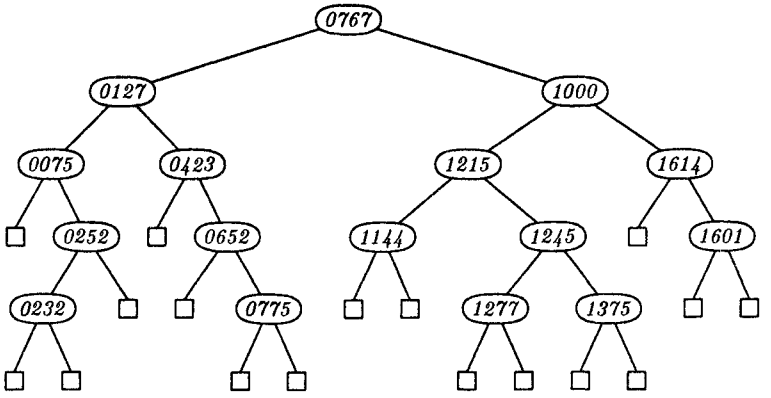


Рис. 35. Дерево случайного цифрового поиска, построенное по алгоритму D.

Для описания соответствующих характеристик деревьев цифрового поиска можно воспользоваться производящей функцией. Пусть на уровне l находится a_l внутренних узлов. Рассмотрим производящую функцию $a(z) = \sum_l a_l z^l$. Допустим, что производящая функция, соответствующая рис. 35, — $a(z) = 1 + 2z + 4z^2 + 5z^3 + 4z^4$. Если на уровне l находится b_l внешних узлов и $b(z) = \sum_l b_l z^l$, то согласно упр. 6.2.1–25 получаем

$$b(z) = 1 + (2z - 1)a(z). \tag{7}$$

Например, $1 + (2z - 1)(1 + 2z + 4z^2 + 5z^3 + 4z^4) = 3z^3 + 6z^4 + 8z^5$. Среднее количество проверок битов при случайном *успешном* поиске равно $a'(1)/a(1)$, так как $a'(1)$ — длина внутреннего пути дерева, а $a(1)$ — количество внутренних узлов. Среднее количество проверок битов при случайном *неудачном* поиске составляет $\sum_l l b_l 2^{-l} = \frac{1}{2}b'(\frac{1}{2}) = a(\frac{1}{2})$, поскольку мы достигаем данного внешнего узла уровня l с вероятностью 2^{-l}. Количество сравнений совпадает с количеством проверок битов (плюс 1 при успешном завершении поиска). Например, на рис. 35 при успешном поиске будет выполнено в среднем $2\frac{9}{16}$ проверок битов и $3\frac{9}{16}$ сравнений, в то время как в случае неудачного поиска обе эти величины будут равны $3\frac{7}{8}$.

Обозначим через $g_N(z)$ усредненную по всем деревьям с N узлами функцию $a(z)$; другими словами, $g_N(z)$ представляет собой сумму $\sum p_T a_T(z)$ по всем бинар-

ным деревьям цифрового поиска T с N внутренними узлами, где $a_T(z)$ — производящая функция для внутренних узлов дерева T, а p_T — вероятность того, что T образуется при вставке N случайных чисел согласно алгоритму D. Тогда среднее количество проверок битов будет равно $g'_N(1)/N$ в случае успешного и $g_N(\frac{1}{2})$ — в случае неудачного поиска.

Можно вычислить $g_N(z)$ при помощи имитации процесса построения дерева следующим образом. Если $a(z)$ представляет собой производящую функцию для дерева с N узлами, можно сформировать из него $N+1$ дерево методом вставки в позицию любого внешнего узла. Эта вставка производится в данный внешний узел уровня l с вероятностью 2^{-l}; следовательно, сумма производящих функций для $N+1$ нового дерева, умноженная на вероятность ее появления, равна $a(z)+b(\frac{1}{2}z) = a(z)+1+(z-1)a(\frac{1}{2}z)$. Усреднение по всем деревьям с N узлами дает

$$g_{N+1}(z) = g_N(z) + 1 + (z-1)g_N(\tfrac{1}{2}z); \qquad g_0(z) = 0. \tag{8}$$

С соответствующей производящей функцией для внешних узлов,

$$h_N(z) = 1 + (2z-1)g_N(z),$$

работать несколько легче в связи с тем, что (8) эквивалентно формуле

$$h_{N+1}(z) = h_N(z) + (2z-1)h_N(\tfrac{1}{2}z); \qquad h_0(z) = 1. \tag{9}$$

Многократно применяя это правило, находим, что

$$\begin{aligned} h_{N+1}(z) &= h_{N-1}(z) + 2(2z-1)h_{N-1}(\tfrac{1}{2}z) + (2z-1)(z-1)h_{N-1}(\tfrac{1}{4}z) \\ &= h_{N-2}(z) + 3(2z-1)h_{N-2}(\tfrac{1}{2}z) + 3(2z-1)(z-1)h_{N-2}(\tfrac{1}{4}z) \\ &\quad + (2z-1)(z-1)(\tfrac{1}{2}z-1)h_{N-2}(\tfrac{1}{8}z) \end{aligned}$$

и т. д. В результате имеем следующее:

$$h_N(z) = \sum_k \binom{N}{k} \prod_{j=0}^{k-1} (2^{1-j}z - 1); \tag{10}$$

$$g_N(z) = \sum_{k\ge 0} \binom{N}{k+1} \prod_{j=0}^{k-1} (2^{-j}z - 1). \tag{11}$$

Например, $g_4(z) = 4 + 6(z-1) + 4(z-1)(\frac{1}{2}z-1) + (z-1)(\frac{1}{2}z-1)(\frac{1}{4}z-1)$. Эти формулы позволяют выразить искомые величины в виде сумм произведений:

$$\bar{C}_N = g'_N(1) = \sum_{k\ge 0} \binom{N}{k+2} \prod_{j=1}^{k} (2^{-j} - 1); \tag{12}$$

$$g_N(\tfrac{1}{2}) = \sum_{k\ge 0} \binom{N}{k+1} \prod_{j=1}^{k} (2^{-j} - 1) = \bar{C}_{N+1} - \bar{C}_N. \tag{13}$$

Совершенно неочевидно, что эти формулы для $\bar{C}_N$ удовлетворяют (6)!

К сожалению, данные выражения непригодны для вычислений или поиска асимптотических зависимостей, поскольку $2^{-j} - 1$ отрицательно; мы получим большие

члены, многие из которых сократятся. Более полезная формула для $\bar{C}_N$ может быть получена в результате применения тождеств из упр. 5.1.1–16:

$$\begin{aligned}
\bar{C}_N &= \Bigl(\prod_{j\ge1}(1-2^{-j})\Bigr)\sum_{k\ge0}\binom{N}{k+2}(-1)^k\prod_{l\ge0}(1-2^{-l-k-1})^{-1}\\
&= \Bigl(\prod_{j\ge1}(1-2^{-j})\Bigr)\sum_{k\ge0}\binom{N}{k+2}(-1)^k\sum_{m\ge0}(2^{-k-1})^m\prod_{r=1}^{m}(1-2^{-r})^{-1}\\
&= \sum_{m\ge0}2^m\Bigl(\sum_k\binom{N}{k}(-2^{-m})^k-1+2^{-m}N\Bigr)\prod_{j\ge0}(1-2^{-j-m-1})\\
&= \sum_{m\ge0}2^m\bigl((1-2^{-m})^N-1+2^{-m}N\bigr)\sum_{n\ge0}(-2^{-m-1})^n\frac{2^{-n(n-1)/2}}{\prod_{r=1}^{n}(1-2^{-r})}.
\end{aligned}\tag{14}$$

На первый взгляд, никакого улучшения по сравнению с (12) не произошло, однако очень большое достоинство полученного результата заключается в том, что сумма по m довольно быстро сходится для любого фиксированного n. Аналогичная ситуация возникает и в случае луча в формулах 5.2.2–(38) и 5.2.2–(39); в самом деле, при рассмотрении только членов с $n = 0$ из (14) получается в точности $N - 1$ плюс количество проверок битов в бинарном луче. Теперь можно получить асимптотическое значение так же, как и ранее (см. упр. 27). [Приведенный вывод базируется, в основном, на подходе, предложенном в работе A. J. Konheim, D. J. Newman, *Discrete Mathematics* **4** (1973), 57–63].

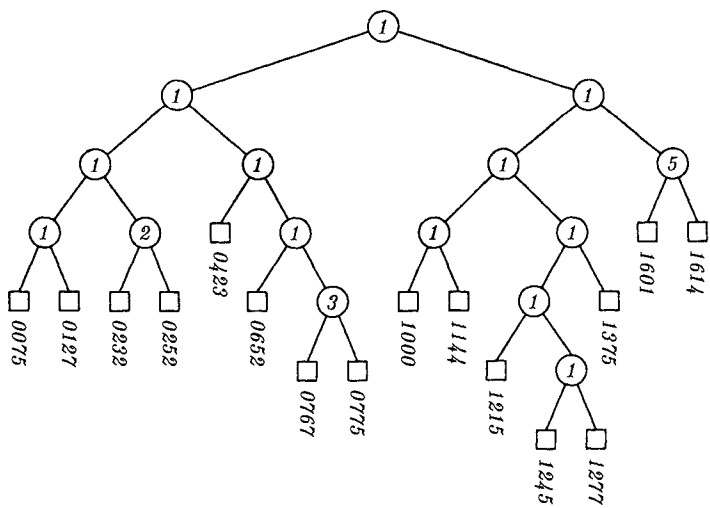


Рис. 36. Дерево, которое строится алгоритмом "Патриция" вместо луча, приведенного на рис. 34.

И, наконец, бросим на метод "Патриция" "математический" взгляд. В таком случае бинарное дерево похоже на соответствующий луч с теми же ключами, но

сжатыми вместе (поля SKIP позволяют устранить однопутевые ветвления), так что всегда имеется ровно $N-1$ внутренних узлов и N внешних узлов. На рис. 36 показано дерево метода "Патриция", соответствующее шестнадцати ключам луча, представленного на рис. 34. Число, показанное в каждом узле ветви, представляет собой значение SKIP; ключи расположены у внешних узлов, хотя они и не присутствуют в дереве явно (в действительности на месте каждого внешнего узла имеется ссылка на внутренний узел, который, в свою очередь, ссылается на массив TEXT). Впрочем, для целей анализа будем считать, что внешние узлы имеют именно такой вид, как показано на рисунке.

Поскольку успешный поиск с использованием метода "Патриция" завершается во внешних узлах, среднее количество проверок битов, выполненных при случайном успешном поиске, будет равно длине внешнего пути, деленной на N. Если сформировать, как и ранее, производящую функцию $b(z)$ для внешних узлов, то эта величина может быть представлена как $b'(1)/b(1)$. *Неудачный* поиск с использованием метода "Патриция" также заканчивается во внешнем узле уровня l с вероятностью 2^{-l}, так что среднее количество проверок битов составляет $\frac{1}{2}b'\left(\frac{1}{2}\right)$. Например, для случая, представленного на рис. 36, мы имеем $b(z) = 3z^3 + 8z^4 + 3z^5 + 2z^6$; таким образом, среднее число проверок битов при успешном поиске равно $4\frac{1}{4}$ и при неудачном — $3\frac{25}{32}$.

Обозначим через $h_n(z)$ "среднюю" производящую функцию $b(z)$, т. е. функцию, усредненную по всем деревьям метода "Патриция" с n внешними узлами и равномерно распределенными ключами. Рекуррентное соотношение

$$h_n(z) = 2^{1-n}\sum_k \binom{n}{k} h_k(z)\big(z + \delta_{kn}(1-z)\big), \quad h_0(z) = 0, \quad h_1(z) = 1, \tag{15}$$

по-видимому, не имеет простого решения; но, к счастью, существует простое рекуррентное соотношение для средней длины внешнего пути $h'_n(1)$, поскольку

$$\begin{aligned} h'_n(1) &= 2^{1-n}\sum_k \binom{n}{k} h'_k(1) + 2^{1-n}\sum_k \binom{n}{k} k(1-\delta_{kn}) \\ &= n - 2^{1-n}n + 2^{1-n}\sum_k \binom{n}{k} h'_k(1). \end{aligned} \tag{16}$$

Так как оно имеет вид (6), для поиска $h'_n(1)$ можно использовать ранее разработанные методы. Оказывается, $h'_n(1)$ ровно на n меньше соответствующего числа проверок битов в случайном бинарном луче. Следовательно, поле SKIP позволяет сэкономить около одной проверки бита при удачном поиске случайных данных (см. упр. 31). Избыточность же типичных реальных данных приведет к еще большей экономии.

Если попытаться найти среднее количество проверок битов в случае *неудачного* поиска методом "Патриция", получится рекуррентное соотношение

$$a_n = 1 + \frac{1}{2^n - 2}\sum_{k<n} \binom{n}{k} a_k \quad \text{для } n \geq 2; \qquad a_0 = a_1 = 0 \tag{17}$$

(здесь $a_n = \frac{1}{2}h'_n\left(\frac{1}{2}\right)$). Это соотношение *не* похоже ни на одно из рассмотренных выше рекуррентных соотношений и не сводится к ним каким-либо более или

менее простым способом. Однако теория преобразований Меллина, приведенная в разделе 5.2.2, обеспечивает возможность работы с такого рода рекуррентными соотношениями. Оказывается, решение (17) содержит числа Бернулли:

$$\frac{na_{n-1}}{2} - n + 2 = \sum_{k=2}^{n-1} \binom{n}{k} \frac{B_k}{2^{k-1} - 1} \quad \text{для } n \geq 2. \tag{18}$$

Эта формула, вероятно, представляет собой самый твердый асимптотический орешек, который был разгрызен в данной книге. Решение, представленное в упр. 34, является поучительным обзором многих ранее встречавшихся задач (с некоторыми нюансами).

Итоги анализа. Следующие результаты сложных математических выкладок этого раздела заслуживают особого внимания.

a) Количество узлов, которые необходимы для хранения N случайных ключей в M-арном луче с разветвлениями, прекращающимися по достижении подфайлов из $\leq s$ ключей, составляет примерно $N/(s \ln M)$. Это приближение справедливо при больших N, малых s и малых M. Поскольку узлы луча содержат M полей ссылок, всего при $s = M$ потребуется около $N/\ln M$ полей ссылок.

b) Количество цифр или символов, проверяемых в ходе случайного поиска, составляет для всех рассмотренных методов примерно $\log_M N$. При $M = 2$ различные методы анализа дают более точные приближения для количества проверок битов.

	Успешный	Неудачный
Поиск по лучу	$\lg N + 1.33275$	$\lg N - 0.10995$
Цифровой поиск по дереву	$\lg N - 1.71665$	$\lg N - 0.27395$
Поиск методом "Патриция"	$\lg N + 0.33275$	$\lg N - 0.31875$

(Эти приближения могут быть выражены фундаментальными математическими постоянными; например, 0.31875 на самом деле означает $(\ln \pi - \gamma)/\ln 2 - 1/2$).

c) "Случайные данные" в нашем случае означают, что M-ичные цифры равномерно распределены, как если бы ключи были действительными числами между 0 и 1, записанными в M-ичной системе счисления. Методы цифрового поиска не зависят от порядка, в котором ключи введены в файл (за исключением алгоритма D, который слабо чувствителен к порядку); однако они весьма чувствительны к распределению цифр. Например, если нулевые биты встречаются гораздо чаще единичных, деревья будут существенно более асимметричными по сравнению с деревьями, полученными для случайных данных (в упр. 5.2.2–53 приведен пример того, что может случиться при таком смещении данных).

УПРАЖНЕНИЯ

1. [*00*] Если дерево имеет листья, то что имеет луч?

2. [*20*] Разработайте алгоритм вставки нового ключа в M-арный луч с использованием соглашений алгоритма T.

3. [*21*] Разработайте алгоритм удаления ключа из M-арного луча с использованием соглашений алгоритма T.

▶ **4.** [*21*] Большинство из 360 элементов, приведенных в табл. 1, представляет собой пустые ссылки. Однако можно сжать таблицу до 49 элементов, перекрыв непустые и пустые

элементы, как показано ниже.

Позиция	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
Элемент		(10)		WAS	THAT	(11)	OF	BE	THE	HIS	WHICH	WITH	THIS		(12)	ON	I	HE	A	OR	(2)	(3)	TO	HAD	

Позиция	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49
Элемент	(4)	BUT	(5)	(6)	FOR	BY	IN	FROM	AND	NOT	(7)	HER	ARE	IS	IT	AS	AT	(8)		HAVE	(9)		YOU	

(Узлы (1), (2), ..., (12) в табл. 1 начинаются соответственно в позициях 20, 19, 3, 14, 1, 17, 1, 7, 3, 20, 18, 4 этой сжатой таблицы.)

Покажите, что, если сжатой таблицей заменить табл. 1, программа T останется работоспособной, хотя и не столь быстрой.

▶ **5.** [*M26*] (Я. Н. Патт (Y. N. Patt).) В деревьях на рис. 31 содержатся буквы каждого семейства, упорядоченные по алфавиту. Такое упорядочение не является необходимым, и, если переупорядочить узлы внутри каждого семейства перед построением дерева типа (2), поиск может ускориться. Какое переупорядочение представленного на рис. 31 дерева приведет к оптимальному с этой точки зрения результату? (Используйте частоты, приведенные на рис. 32, и найдите "лес", который позволит минимизировать время успешного поиска, когда он представлен в виде бинарного дерева.)

6. [*15*] Какое дерево цифрового поиска получится при вставке 15 четырехбитовых бинарных ключей 0001, 0010, 0011, ..., 1111 в порядке возрастания согласно алгоритму D? (Начните с ключа 0001 в корневом узле и выполните 14 вставок).

▶ **7.** [*M26*] Если 15 ключей из упр. 6 вставлены в другом порядке, может быть построено другое дерево. Какая перестановка этих ключей из всех 15! возможных будет *наихудшей* в том смысле, что она породит дерево с наибольшей длиной внутреннего пути?

8. [*20*] Рассмотрим следующие изменения в алгоритме D, ведущие к исключению из него переменной K': заменим "K'" на "K" на шаге D2 и удалим операцию "$K' \leftarrow K$" на шаге D1. Будет ли полученный в результате этих действий алгоритм корректным, и можно ли с его помощью выполнять поиск и вставку?

9. [*21*] Напишите MIX-программу для алгоритма D и сравните ее с программой 6.2.2T. Можете использовать бинарные операции, такие как SLB (бинарный сдвиг AX влево), JAE (переход при четном A) и др.; возможно, вам поможет упр. 8.

10. [*23*] Дан файл, все ключи которого представляют собой n-битовые двоичные числа, и дан аргумент поиска $K = b_1 b_2 \ldots b_n$. Предположим, что необходимо найти максимальное значение k, при котором в файле будет содержаться ключ, начинающийся с $b_1 b_2 \ldots b_k$. Как эффективно решить поставленную задачу, если файл представлен в виде

a) бинарного дерева поиска (алгоритм 6.2.2T);

b) бинарного луча (алгоритм T);

c) бинарного дерева цифрового поиска (алгоритм D)?

11. [*21*] Можно ли использовать неизмененный алгоритм 6.2.2D для удаления узла из дерева цифрового поиска?

12. [*25*] Будет ли случайным дерево, полученное путем удаления случайного элемента из случайного дерева цифрового поиска, которое построено с помощью алгоритма D? (См. упр. 11 и теорему 6.2.2H.)

13. [*20*] (*M-арный цифровой поиск.*) Объясните, как можно объединить алгоритмы T и D в один обобщенный алгоритм, при $M = 2$ представляющий собой алгоритм D. Какие изменения следует внести в табл. 1 для использования алгоритма при $M = 30$?

▶ **14.** [*25*] Напишите эффективный алгоритм, который может быть выполнен сразу после успешного окончания алгоритма P для нахождения *всех* мест, в которых TEXT содержит K.

15. [*28*] Разработайте эффективный алгоритм, который может применяться для построения дерева, используемого методом "Патриция", или для вставки новой ссылки на TEXT в существующее дерево. Ваш алгоритм вставки должен обращаться к массиву TEXT не более двух раз.

16. [*22*] Почему в алгоритме "Патриция" требуется, чтобы один ключ не служил началом другого?

17. [*M25*] Как выразить решение рекуррентного соотношения

$$x_0 = x_1 = 0, \qquad x_n = a_n + m^{1-n} \sum_k \binom{m}{k} (m-1)^{n-k} x_k, \qquad n \geq 2,$$

с помощью биномиальных преобразований, обобщая метод упр. 5.2.2–36?

18. [*M21*] Используя результат упр. 17, выразите решения уравнений (4) и (5) через функции U_n и V_n, аналогичные определенным в упр. 5.2.2–38.

19. [*HM23*] Найдите асимптотическое значение функции

$$K(n, s, m) = \sum_{k \geq 2} \binom{n}{k} \binom{k}{s} \frac{(-1)^k}{m^{k-1} - 1}$$

с точностью $O(1)$ при $n \to \infty$ для фиксированных значений $s \geq 0$ и $m > 1$. [Случай для $s = 0$ рассматривался в упр. 5.2.2–50, а случай для $s = 1$, $m = 2$ — в упр. 5.2.2–48.]

▶ **20.** [*M30*] Рассмотрим M-арную "лучевую" память, в которой используется последовательный поиск по достижении поддерева из s или меньшего количества лучей (алгоритм T представляет собой частный случай при $s = 1$). Примените результаты предыдущих упражнений для анализа

a) среднего количества узлов луча;
b) среднего количества проверок цифр или символов при успешном поиске;
c) среднего количества сравнений, выполняемых при успешном поиске.

Сформулируйте ответы на вопросы в виде асимптотических формул при $N \to \infty$ для фиксированных M и s; ответ для (a) должен быть дан с точностью до $O(1)$, а для (b) и (c) — до $O(N^{-1})$. [При $M = 2$ этот анализ применим к модифицированному методу обменной поразрядной сортировки, в котором подфайлы размером $\leq s$ сортируются посредством вставок.]

21. [*M25*] Сколько узлов случайного M-арного луча с N ключами имеют в качестве нулевой компоненты пустую ссылку? (Например, 9 из 12 узлов в табл. 1 имеют пустую ссылку в позиции "␣". "Случайность" в данном упражнении, как обычно, означает, что цифры ключей равномерно распределены между 0 и $M - 1$.)

22. [*M25*] Сколько узлов находится в случайном M-арном луче с N ключами на уровне l ($l = 0, 1, 2, \ldots$)?

23. [*M26*] Сколько проверок цифр выполняется в среднем в процессе *не*успешного поиска в M-арном луче, содержащем N случайных ключей?

24. [*M30*] Рассмотрите M-арный луч, представленный в виде леса (см. рис. 31). Найдите точные и асимптотические выражения для

a) среднего количества узлов в лесу;

b) среднего количества присвоений "P ← RLINK(P)" в процессе случайного успешного поиска.

▶ **25.** [*M24*] Математический вывод асимптотических значений в этом разделе весьма сложен (использовалась даже теория функций комплексного переменного). Дело в том, что мы не хотели ограничиться одним членом асимптотической формулы, а вывод второго члена действительно сложен. Назначение данного упражнения — показать, что элементарных методов достаточно для вывода некоторых (пусть и ослабленных) результатов.

a) Докажите по индукции, что решение (4) удовлетворяет неравенству $A_N \le M(N-1)/(M-1)$.

b) Пусть $D_N = C_N - NH_{N-1}/\ln M$, где C_N определяется формулой (5). Докажите, что $D_N = O(N)$; следовательно, $C_N = N\log_M N + O(N)$. [*Указание.* Используйте (a) и теорему 1.2.7A.]

26. [*23*] Определите значение бесконечного произведения

$$(1-\tfrac{1}{2})(1-\tfrac{1}{4})(1-\tfrac{1}{8})(1-\tfrac{1}{16})\ldots$$

с точностью до пяти значащих цифр непосредственным вычислением. [*Указание.* См. упр. 5.1.1–16.]

27. [*HM31*] Чему с точностью до $O(1)$ равно асимптотическое значение $\bar{C}_N$ из (14)?

28. [*HM26*] Найдите асимптотическое среднее количество проверок цифр при выполнении в случайном M-арном дереве цифрового поиска при $M \ge 2$. Рассмотрите случай как успешного, так и неудачного поиска; дайте ответ с точностью до $O(N^{-1})$.

29. [*HM40*] Чему равно асимптотическое среднее количество узлов в M-арном дереве цифрового поиска, все ссылки которых пусты? (Можно было бы сэкономить память, исключая такие узлы; см. упр. 13.)

30. [*M24*] Покажите, что производящая функция алгоритма "Патриция" $h_n(z)$, определенная в (15), может быть выражена в совершенно ужасном виде:

$$n\sum_{m\ge1} z^m\left(\sum_{\substack{a_1+\cdots+a_m=n-1\\ a_1,\ldots,a_m\ge1}}\binom{n-1}{a_1,\ldots,a_m}\frac{1}{(2^{a_1}-1)(2^{a_1+a_2}-1)\ldots(2^{a_1+\cdots+a_m}-1)}\right).$$

(Таким образом, если бы имелась простая формула для $h_n(z)$, можно было бы упростить это громоздкое выражение.)

31. [*M21*] Решите рекуррентное соотношение (16).

32. [*M21*] Чему равно среднее значение суммы всех полей SKIP в случайном дереве алгоритма "Патриция" с $N-1$ внутренним узлом?

33. [*M30*] Докажите, что (18) представляет собой решение рекуррентного соотношения (17). [*Указание.* Рассмотрите производящую функцию $A(z) = \sum_{n\ge0} a_n z^n/n!$.]

34. [*HM40*] Назначение данного упражнения — поиск асимптотического поведения формулы (18).

a) Докажите, что при $n \ge 2$

$$\frac{1}{n}\sum_{2\le k<n}\binom{n}{k}\frac{B_k}{2^{k-1}-1} = \sum_{j\ge1}\left(\frac{1^{n-1}+2^{n-1}+\cdots+(2^j-1)^{n-1}}{2^{j(n-1)}} - \frac{2^j}{n} + \frac{1}{2}\right).$$

b) Покажите, что слагаемые в правой части (a) приближенно равны $1/(e^x-1)-1/x+1/2$, где $x = n/2^j$; получающаяся при этом сумма отличается от первоначальной на $O(n^{-1})$.

c) Покажите, что

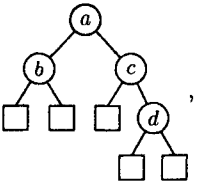
$$\frac{1}{e^x-1}-\frac{1}{x}+\frac{1}{2}=\frac{1}{2\pi i}\int_{-\frac{1}{2}-i\infty}^{-\frac{1}{2}+i\infty}\zeta(z)\Gamma(z)\,x^{-z}dz \qquad \text{для действительного } x>0.$$

d) Следовательно, рассматриваемая сумма равна

$$\frac{1}{2\pi i}\int_{-\frac{1}{2}-i\infty}^{-\frac{1}{2}+i\infty}\frac{\zeta(z)\Gamma(z)n^{-z}}{2^{-z}-1}\,dz+O(n^{-1}).$$

Оцените этот интеграл.

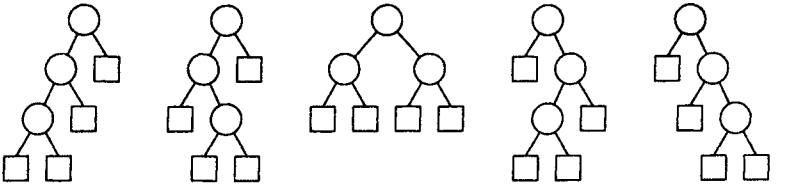
▶ **35.** [*M20*] Какова вероятность того, что дерево алгоритма "Патриция" с пятью ключами имеет вид



,

причем в полях SKIP содержатся a, b, c, d, как показано на рисунке? (Считаем, что ключи имеют независимые случайные биты; ответ должен иметь вид функции от a, b, c и d.)

36. [*M25*] Имеется пять бинарных деревьев с тремя внутренними узлами в каждом. Если рассмотреть, как часто каждое из них служит деревом поиска в различных алгоритмах при случайных данных, то получатся следующие различные вероятности.



Поиск по дереву (алгоритм 6.2.2T)	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{3}$	$\frac{1}{6}$	$\frac{1}{6}$
Цифровой поиск по дереву (алгоритм D)	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{2}$	$\frac{1}{8}$	$\frac{1}{8}$
Метод "Патриция" (алгоритм P)	$\frac{1}{7}$	$\frac{1}{7}$	$\frac{3}{7}$	$\frac{1}{7}$	$\frac{1}{7}$

(Заметим, что дерево цифрового поиска будет сбалансировано чаще других.) В упр. 6.2.2–5 приведена формула для вероятности в случае поиска по дереву: $\prod(1/s(x))$, где произведение берется по всем внутренним узлам x, а $s(x)$ — число внутренних узлов в поддереве, корнем которого является x. Найдите аналогичные формулы для вероятности в случае (a) алгоритма D и (b) алгоритма P.

▶ **37.** [*M22*] Рассмотрите бинарное дерево, на уровне l которого имеется b_l внешних узлов. В тексте указывалось, что время неудачного поиска в деревьях цифрового поиска не связано с длиной внешнего пути $\sum lb_l$ непосредственно, а пропорционально *модифицированной длине внешнего пути* $\sum lb_l2^{-l}$. Докажите или опровергните следующее утверждение: среди всех деревьев с N внешними узлами наименьшую модифицированную длину внешнего пути имеет то дерево, все внешние узлы которого расположены не более чем на двух смежных уровнях (см. упр. 5.3.1–20).

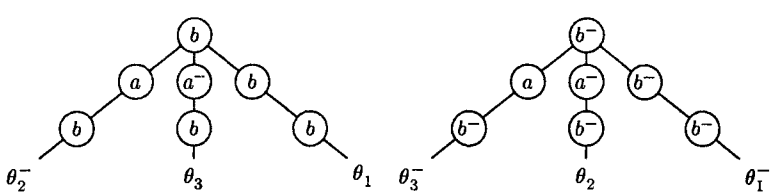
38. [*M40*] Разработайте алгоритм поиска дерева с n узлами, имеющего минимальное значение величины $\alpha\cdot$(длина внутреннего пути)$+\beta\cdot$(модифицированная длина внешнего пути), где α и β — некоторые числа (см. упр. 37).

39. [*M43*] Разработайте алгоритм поиска оптимальных деревьев цифрового поиска, аналогичных оптимальным бинарным деревьям поиска, которые рассматривались в разделе 6.2.2.

▶ **40.** [*25*] Пусть $a_0\,a_1\,a_2\ldots$ — это периодическая бинарная последовательность, в которой $a_{N+k} = a_k$ для всех $k \geq 0$. Покажите, что любая фиксированная последовательность такого типа может быть представлена в $O(N)$ ячейках памяти так, что следующая операция может быть выполнена за $O(N)$ шагов. Определите для данной цепочки битов $b_0\,b_1\ldots b_{n-1}$, сколько раз она встречается в периоде (т. е. найдите, сколько существует таких значений p, при которых $0 \leq p < N$ и $b_k = a_{p+k}$ для $0 \leq k < n$). Предполагается, что каждая ячейка памяти достаточно велика, чтобы содержать любое целое число от 0 до N. [*Указание.* См. упр. 14.]

41. [*HM28*] (Приложение к теории групп.) Пусть G — свободная группа над символами $\{a_1,\ldots,a_n\}$, т. е. множество всех строк $\alpha = b_1\ldots b_r$, где каждый b_i представляет собой либо a_j, либо a_j^- и не имеется смежных пар $a_j a_j^-$ или $a_j^- a_j$. Обратным к α элементом является $b_r^-\ldots b_1^-$. Перемножим две такие строки путем их конкатенации и сокращения смежных обратных пар. Пусть H — подгруппа группы G, порожденная строками $\{\beta_1,\ldots,\beta_p\}$, т. е. множество всех элементов G, которые могут быть записаны в виде произведений β и обратных им элементов. Согласно широкоизвестной теореме Якоба Нильсена (Jakob Nielsen) [см. Marshall Hall, *The Theory of Groups* (New York: Macmillan, 1959), Ch. 7] всегда можно найти образующие $\theta_1,\ldots,\theta_m$ для H, $m \leq p$, обладающие тем свойством, что средний символ θ_i (или, по крайней мере, один из центральных символов θ_i, если его длина четна) никогда не сокращается в выражениях $\theta_i\theta_j^e$ или $\theta_j^e\theta_i$, $e = \pm1$, за исключением случая $j = i$ и $e = -1$. Из этого свойства следует, что существует простой алгоритм проверки принадлежности некоторого элемента G подгруппе H: запишем $2m$ ключей $\theta_1,\ldots,\theta_m$, $\theta_1^-,\ldots,\theta_m^-$ в дерево, ориентированное на поиск по символам, с помощью $2n$ букв $a_1,\ldots,a_n$, $a_1^-,\ldots,a_n^-$. Пусть $\alpha = b_1\ldots b_r$ — данный элемент из G; если $r = 0$, то α, разумеется, принадлежит H. В противном случае ищем α, определяя самый длинный префикс $b_1\ldots b_k$, который соответствует ключу. Если имеется несколько ключей, начинающихся с $b_1\ldots b_k$, то α не принадлежит H. Иначе обозначим этот единственный ключ через $b_1\ldots b_k c_1\ldots c_l = \theta_i^e$ и заменим α ключом $\theta_i^{-e}\alpha = c_l^-\ldots c_1^- b_{k+1}\ldots b_r$. Если новое значение α длиннее старого (т. е. если $l > k$), α не принадлежит H; в противном случае повторяем процесс с новым значением α. Из свойства Нильсена вытекает конечность этого алгоритма. Если α сводится к пустой строке, можно восстановить исходное представление α в виде произведений θ.

Например, пусть $\{\theta_1,\theta_2,\theta_3\} = \{bbb,\ b^-a^-b^-,\ ba^-b\}$ и $\alpha = bbabaab$. Лес



вместе с описанным алгоритмом позволяет вывести, что $\alpha = \theta_1\theta_3^-\theta_1\theta_3^-\theta_2^-$. Реализуйте этот алгоритм, используя в качестве входных данных вашей программы θ_i.

42. [*23*] (*Сжатие спереди и сзади.*) Если множество бинарных ключей используется в качестве индекса для разделения большого файла, можно не хранить полные ключи. Например, 16 ключей, показанных на рис. 34, могут быть урезаны справа так, чтобы

оставалось достаточное количество цифр для их однозначной идентификации: 0000, 0001, 00100, 00101, 010, ..., 1110001. Такие урезанные ключи могут использоваться для разделения файла на 17 частей, и, например, в пятой части могут содержаться все ключи, начинающиеся с 0011 или 010, а в последней части — ключи, начинающиеся с 111001, 11101 или 1111. Урезанные ключи можно представить в более компактной форме, если опустить все лидирующие цифры, совпадающие с цифрами предыдущего ключа: 0000, ◇◇◇1, ◇◇100, ◇◇◇◇1, ◇10, ..., ◇◇◇◇◇◇1. Бит, следующий за символом "◇", всегда равен 1, поэтому он также может быть опущен. В большом файле будет содержаться много символов "◇", и мы должны хранить только их количество и значения последующих битов.

Покажите, что общее количество битов в сжатом файле с исключенными символами "◇" и последующим одним битом, всегда равно количеству узлов в бинарном луче с этими ключами.

(Следовательно, среднее общее количество таких битов во всем индексе составляет около $N/\ln 2$, 1.44 бит на ключ. Этот способ сжатия был показан автору Э. Геллером (A. Heller) и Р. Л. Джонсеном (R. L. Johnsen). Возможно и дальнейшее сжатие, поскольку нам необходимо только представление структуры луча; см. теорему 2.3.1A.)

43. [*HM42*] Проанализируйте высоту случайного M-арного луча, имеющего N ключей и параметр обрезки s, как в упр. 20 (когда $s = 1$, эта величина представляет собой длину самого длинного общего префикса случайных слов длиной N в M-арном алфавите).

▶ **44.** [*30*] (Дж. Л. Бентли (J. L. Bentley) и Р. Седгевик (R. Sedgewick).) Исследуйте тернарное представление лучей, при котором левая и правая ссылки соответствуют горизонтальным ветвям (2), в то время как средняя ссылка соответствует ветви, идущей вниз.

▶ **45.** [*M25*] Если семь ключей, которые показаны на рис. 33, вставлены в случайном порядке с помощью алгоритма, описанного в упр. 15, то чему равна вероятность получения показанного дерева?

6.4. ХЕШИРОВАНИЕ

До сих пор мы рассматривали методы поиска, основанные на сравнении данного аргумента K с ключами в таблице или на использовании его цифр для управления процессом разветвления. Третий путь заключается в том, чтобы отказаться от поиска по данным и выполнить арифметические действия над K, позволяющие вычислить некоторую функцию $f(K)$. Последняя укажет адрес в таблице, где хранится K и связанная с ним информация.

Например, рассмотрим хорошо известное нам множество из 31 английского слова, которое приводилось в разделах 6.2.2 и 6.3. В табл. 1 показана небольшая MIX-программа, взаимно однозначно преобразующая 31 ключ в числа $f(K)$ между -10 и 30. Если сравнить этот метод с MIX-программами для других рассматривавшихся методов (например, для бинарного поиска, оптимального поиска по дереву, "лучевого" поиска, цифрового поиска по дереву), то можно обнаружить, что новый способ превосходит старые как с точки зрения быстродействия, так и с точки зрения количества используемой памяти (за исключением бинарного поиска, для которого необходим несколько меньший объем памяти). В самом деле, среднее время успешного поиска с помощью программы из табл. 1 составляет около $17.8u$ и требует для хранения 31 ключа таблицу из 41 элемента.

К сожалению, поиск таких функций $f(K)$ — довольно сложная задача. Имеется $41^{31} \approx 10^{50}$ возможных функций, отображающих 31-элементное множество в 41-элементное; однако только $41 \cdot 40 \cdot \ldots \cdot 11 = 41!/10! \approx 10^{43}$ из них дают различные значения для различных аргументов; следовательно, только одна из десяти миллионов функций подходит для достижения нашей цели.

Функции, дающие неповторяющиеся значения, встречаются на удивление редко (даже для больших таблиц). Например, в соответствии со знаменитым "парадоксом дней рождения" получается, что в компании из 23 человек более вероятно совпадение дней рождения у двух из них, чем то, что у всех дни рождения окажутся различными. Иными словами, если выбрать случайную функцию, отображающую 23 ключа в таблицу размером 365, вероятность того, что никакие два ключа не будут отображены в одно и то же место таблицы, составляет 0.4927, т. е. менее половины. Скептики могут проверить парадокс на ближайшей многолюдной вечеринке*. [Парадокс дней рождения неформально обсуждался математиками в 30-е годы; происхождение этого парадокса неизвестно. См. I. J. Good, *Probability and the Weighing of Evidence* (Griffin, 1950), 38, а также R. von Mises, *İstanbul Üniversitesi Fen Fakültesi Mecmuasi* **4** (1939), 145–163; W. Feller, *An Introduction to Probability Theory* (New York: Wiley, 1950), Section 2.3.]

С другой стороны, использованный в табл. 1 подход весьма гибок [см. M. Greniewski and W. Turski, *CACM* **6** (1963), 322–323], и для таблиц небольшого размера подходящая функция может быть найдена за день-другой. Решать такого рода головоломки весьма занятно! Вопросы поиска подходящих функций рассматривались многими исследователями, включая, например, R. Sprugnoli, *CACM* **20** (1977), 841–850, **22** (1979), 104, 553; R. J. Cichelli, *CACM* **23** (1980), 17–19; T. J. Sager, *CACM* **28** (1985), 523–532, **29** (1986), 557; B. S. Majewski, N. C. Wormald, G. Havas

---

* Любители математики могут обратиться за подробностями, например, к книге У. Болла и Г. Коксетера *Математические эссе и развлечения* (М.: Мир, 1986. — 474 с.). Скептикам-нелюдимам рекомендуется вспомнить свой класс в школе или группу в институте. — *Прим. перев.*

Таблица 1

ВЗАИМНО ОДНОЗНАЧНОЕ ПРЕОБРАЗОВАНИЕ МНОЖЕСТВА КЛЮЧЕЙ В АДРЕСА

	Команда		A	AND	ARE	AS	AT	BE	BUT	BY	FOR	FROM	HAD	HAVE	HE	HER
	LD1N	K(1:1)	−1	−1	−1	−1	−1	−2	−2	−2	−6	−6	−8	−8	−8	−8
	LD2	K(2:2)	−1	−1	−1	−1	−1	−2	−2	−2	−6	−6	−8	−8	−8	−8
	INC1	-8,2	−9	6	10	13	14	−5	14	18	2	5	−15	−15	−11	−11
	J1P	*+2	−9	6	10	13	14	−5	14	18	2	5	−15	−15	−11	−11
	INC1	16,2	7	.	.	.	.	16	.	.	.	.	2	2	10	10
	LD2	K(3,3)	7	6	10	13	14	16	14	18	2	5	2	2	10	10
	J2Z	9F	7	6	10	13	14	16	14	18	2	5	2	2	10	10
	INC1	-28,2	.	−18	−13	.	.	.	9	.	−7	−7	−22	−1	.	−1
	J1P	9F	.	−18	−13	.	.	.	9	.	−7	−7	−22	−1	.	−1
	INC1	11,2	.	−3	3	.	.	.	.	.	23	20	−7	35	.	.
	LDA	K(4:4)	.	−3	3	.	.	.	.	.	23	20	−7	35	.	.
	JAZ	9F	.	−3	3	.	.	.	.	.	23	20	−7	35	.	.
	DEC1	-5,2	.	.	.	.	.	.	.	.	.	9	.	15	.	.
	J1N	9F	.	.	.	.	.	.	.	.	.	9	.	15	.	.
	INC1	10	.	.	.	.	.	.	.	.	.	19	.	25	.	.
9H	LDA	K	7	−3	3	13	14	16	9	18	23	19	−7	25	10	1
	CMPA	TABLE,1	7	−3	3	13	14	16	9	18	23	19	−7	25	10	1
	JNE	FAILURE	7	−3	3	13	14	16	9	18	23	19	−7	25	10	1

and Z. J. Czech, *Comp. J.* **39** (1996), 547–554; Czech, Havas, and Majewski, *Theoretical Comp. Sci.* **182** (1997), 1–143. См. также статью J. Körner and K. Marton, *Europ. J. Combinatorics* **9** (1988), 523–530, о теоретических ограничениях идеальных хеш-функций.

Конечно, этот метод имеет очень существенный недостаток, заключающийся в том, что содержимое таблицы должно быть заранее известно. Добавив всего один ключ, можно все испортить, и придется начинать всю работу с нуля. Впрочем, можно получить гораздо более гибкий метод, отбрасывая требование однозначности и используя специальные методы разрешения возникающих неоднозначных ситуаций после вычисления $f(K)$.

Итак, мы вплотную приблизились к популярному классу методов поиска с общим названием *хеширование* (*hashing*) или *рассеянная память* (*scatter storage*). Глагол "to hash" означает "крошить, рубить и делать из этого месиво"*. Идея хеширования состоит в использовании некоторой частичной информации, полученной из ключа, в качестве основы поиска. Мы вычисляем *хеш-адрес* $h(K)$ и используем его для проведения поиска.

Парадокс дней рождения предостерегает нас, что, вероятно, найдутся различные ключи $K_i \neq K_j$, имеющие одинаковое значение хеш-функции $h(K_i) = h(K_j)$. Такое событие именуется *коллизией* (*collision*); для разрешения коллизий разработаны некоторые интересные подходы. Для использования хеширования программист должен решить две практически независимые задачи — выбрать хеш-функцию

* В связи с этим был очень велик соблазн использовать для термина *хеширование* русское слово *окрошка*. — *Прим. перев.*

HIS	I	IN	IS	IT	NOT	OF	ON	OR	THAT	THE	THIS	TO	WAS	WHICH	WITH	YOU
Содержимое rI1 после выполнения команды с определенным ключом K																
−8	−9	−9	−9	−9	−15	−16	−16	−16	−23	−23	−23	−23	−26	−26	−26	−28
−8	−9	−9	−9	−9	−15	−16	−16	−16	−23	−23	−23	−23	−26	−26	−26	−28
−7	−17	−2	5	6	−7	−18	−9	−5	−23	−23	−23	−15	−33	−26	−25	−20
−7	−17	−2	5	6	−7	−18	−9	−5	−23	−23	−23	−15	−33	−26	−25	−20
18	−1	29	.	.	25	4	22	30	1	1	1	17	−16	−2	0	12
18	−1	29	5	6	25	4	22	30	1	1	1	17	−16	−2	0	12
18	−1	29	5	6	25	4	22	30	1	1	1	17	−16	−2	0	12
12	.	.	.	.	20	.	.	.	−26	−22	−18	.	−22	−21	−5	8
12	.	.	.	.	20	.	.	.	−26	−22	−18	.	−22	−21	−5	8
.	.	.	.	.	.	.	.	.	−14	−6	2	.	11	−1	29	.
.	.	.	.	.	.	.	.	.	−14	−6	2	.	11	−1	29	.
.	.	.	.	.	.	.	.	.	−14	−6	2	.	11	−1	29	.
.	.	.	.	.	.	.	.	.	−10	.	−2	.	.	−5	11	.
.	.	.	.	.	.	.	.	.	−10	.	−2	.	.	−5	11	.
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	21	.
12	−1	29	5	6	20	4	22	30	−10	−6	−2	17	11	−5	21	8
12	−1	29	5	6	20	4	22	30	−10	−6	−2	17	11	−5	21	8
12	−1	29	5	6	20	4	22	30	−10	−6	−2	17	11	−5	21	8

$h(K)$ и способ разрешения коллизий. В этом разделе будут рассмотрены обе эти задачи.

Хеш-функции. Для определенности в данном разделе предполагается, что хеш-функция имеет не более M различных значений, причем для всех ключей K

$$0 \leq h(K) < M. \tag{1}$$

На практике ключи в реальном файле зачастую избыточны, а потому мы должны быть очень внимательны при поиске хеш-функции, чтобы не допустить создания кластеров из близких ключей с целью уменьшения количества коллизий.

Теоретически невозможно определить хеш-функцию так, чтобы она создавала случайные данные из реальных неслучайных файлов. Однако на практике не так уж сложно создать неплохую имитацию случайных данных с помощью простых арифметических действий, как обсуждалось в главе 3. В действительности зачастую можно достичь даже лучших результатов, находя и используя неслучайные свойства данных для построения хеш-функций с минимальным количеством коллизий (меньшим, чем при истинно случайных данных).

Рассмотрим, например, десятизначные ключи на десятичном компьютере. Напрашивается следующий способ выбора хеш-функции: положить, скажем, $M = 1000$ и в качестве $h(K)$ использовать три цифры, выбранные около средины двадцатизначного произведения $K \times K$. Казалось бы, этот метод должен давать довольно равномерное распределение значений между 000 и 999 с низкой вероятностью коллизий, однако эксперименты с реальными данными показывают, что такой метод "середины квадрата" неплох при условии, что ключи не имеют большого количества

нулей слева или справа. Подобно тому, как в главе 3 метод середины квадрата оказался не самым хорошим датчиком случайных чисел, но для него нашлись альтернативные методы, так и в нашем случае имеются более простые и надежные способы хеширования.

Многочисленные тесты показали хорошую работу двух основных типов хеш-функций, один из которых основан на делении, а другой — на умножении.

Метод деления весьма прост; мы просто используем остаток от деления на M:

$$h(K) = K \bmod M. \tag{2}$$

В этом случае очевидно, что, например, при четном M значение $h(K)$ будет четным при четном K и нечетным — при нечетном, что приведет к значительному смещению данных во многих файлах. Еще хуже обстоят дела, если M представляет собой степень основания счисления компьютера, поскольку при этом $K \bmod M$ представляет собой несколько цифр числа K, расположенных справа, и не зависит от остальных цифр. Точно так же можно показать, что M не должно быть кратно трем, поскольку при буквенных ключах два из них, отличающиеся только перестановкой букв, могут давать числовые значения с разностью, кратной 3 (это происходит, поскольку $2^{2n} \bmod 3 = 1$ и $10^n \bmod 3 = 1$). В целом, следует избегать значений M, делящих $r^k \pm a$, где k и a — небольшие числа, а r — "основание системы счисления" набора используемых алфавитно-цифровых символов (обычно $r = 64$, 256 или 100), так как остаток от деления по модулю на такие значения M зачастую представляет простую суперпозицию цифр ключа. Приведенные рассуждения приводят к мысли, что лучше всего *использовать в качестве M простое число*, такое, что $r^k \not\equiv \pm a$ (по модулю M) при небольших k и a. В большинстве случаев подобный выбор вполне удовлетворителен.

Например, для MIX-компьютера можно выбрать $M = 1009$, вычисляя $h(K)$ при помощи следующих операций.

```
LDX  K        rX ← K.
ENTA 0        rA ← 0.                    (3)
DIV  =1009=   rX ← K mod 1009.
```

Мультипликативная схема хеширования также просто реализуется, однако сложнее описывается, поскольку необходимо представить, что мы работаем с дробями, а не с целыми числами. Пусть w — размер машинного слова* (что в случае MIX-компьютера обычно составляет 10^{10} или 2^{30}). Целое число A можно рассматривать как дробь A/w, если представить, что разделяющая точка (точка, разделяющая целую и дробную части числа в различных системах счисления, например десятичная точка) расположена слева от числа. Метод состоит в выборе некоторой целой константы A, взаимно простой с w, после чего можно положить

$$h(K) = \left\lfloor M \left(\left(\frac{A}{w} K \right) \bmod 1 \right) \right\rfloor. \tag{4}$$

В этом случае обычно на двоичном компьютере в качестве M используется степень двойки, так что $h(K)$ состоит из старших битов правой половины произведения AK.

---

* Напомним, что Д. Кнут понимает под *размером* машинного слова не количество содержащихся в нем битов, а максимальное количество представляемых им значений. Так, размер 32-битового слова IBM PC по Кнуту составляет $2^{32} = 4\,294\,967\,296$. — *Прим. перев.*

Для бинарного MIX-компьютера, полагая, что $M = 2^m$, хеш-функция вычисляется так.

```
LDA  K   rA ← K.
MUL  A   rAX ← AK.
ENTA 0   rAX ← AK mod w.                    (5)
SLB  m   Сдвиг rAX на m бит влево.
```

Вычисленное значение $h(K)$ помещается в регистр A. Поскольку MIX достаточно медленно производит операции умножения и сдвига, программы (3) и (5) работают одинаковое время; однако на многих компьютерах умножение выполняется значительно быстрее деления.

По сути, предложенный метод представляет собой обобщение метода (3), поскольку можно, например, в качестве A использовать приближение $w/1009$; умножение на обратную величину зачастую происходит быстрее деления. Технология (5) практически совпадает с методом середины квадрата, но с одним важным отличием: в дальнейшем мы увидим, что умножение на подходящую константу имеет много полезных свойств.

Одна из привлекательных черт мультипликативной схемы заключается в отсутствии потери информации в (5); мы в состоянии восстановить значение K по содержимому регистра rAX по окончании работы (5). Дело в том, что A и w взаимно просты и при помощи алгоритма Евклида можно найти константу A', такую, что $AA' \bmod w = 1$; отсюда следует, что $K = \bigl(A'(AK \bmod w)\bigr) \bmod w$. Другими словами, если обозначить через $f(K)$ содержимое регистра X непосредственно перед выполнением команды SLB в (5), то

$$K_1 \neq K_2 \qquad \text{повлечет} \qquad f(K_1) \neq f(K_2). \tag{6}$$

Конечно, $f(K)$ принимает значения в диапазоне от 0 до $w-1$, поэтому ее нельзя считать сколь-нибудь хорошей хеш-функцией. Однако она может быть весьма полезной в качестве *рассеивающей функции* (*scrambling function*), т. е. функции, удовлетворяющей (6) и обычно рандомизирующей ключи. Такая функция может эффективно использоваться и в связи с алгоритмами поиска по дереву из раздела 6.2.2 (если порядок ключей не имеет значения), поскольку она предотвращает возможность построения вырожденного дерева в случае поступления ключей в порядке возрастания (см. упр. 6.2.2–10). Рассеивающая функция может быть применена и для цифрового поиска по дереву из раздела 6.3 при смещении битов действительных ключей.

Другое свойство мультипликативного хеш-метода состоит в том, что он хорошо использует то, что реальные файлы неслучайны. Например, часто множества ключей представляют собой арифметические прогрессии, когда в файле содержатся ключи $\{K, K+d, K+2d, \ldots, K+td\}$. Например, рассмотрим имена типа {PART1, PART2, PART3} или {TYPEA, TYPEB, TYPEC}. Мультипликативный хеш-метод преобразует арифметическую прогрессию в приближенно арифметическую прогрессию $h(K)$, $h(K+d)$, $h(K+2d)$, ... различных хеш-значений, уменьшая число коллизий по сравнению со случайной ситуацией. Метод деления обладает тем же свойством.

На рис. 37 проиллюстрирован этот аспект мультипликативного хеширования в интересном частном случае. Предположим, что A/w приближенно равно золотому сечению $\phi^{-1} = \bigl(\sqrt{5}-1\bigr)/2 \approx 0.6180339887$; при этом последовательность значений

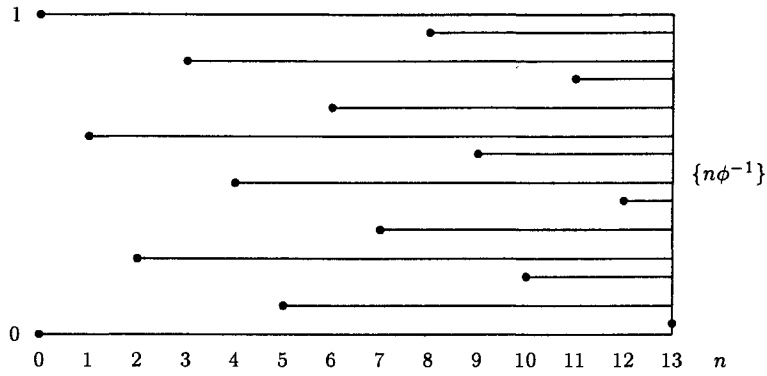


Рис. 37. Хеширование Фибоначчи.

$h(K)$, $h(K+1)$, $h(K+2)$, ... ведет себя точно так же, как последовательность хеш-значений $h(0)$, $h(1)$, $h(2)$, ..., так что естественным становится следующий эксперимент: на отрезке $[0\,..\,1]$ последовательно отмечаем точки $\{\phi^{-1}\}$, $\{2\phi^{-1}\}$, $\{3\phi^{-1}\}$, ..., где $\{x\}$ означает дробную часть числа x (т. е. $x - \lfloor x \rfloor$ или $x \bmod 1$). Как показано на рис. 37, эти точки достаточно хорошо отделены одна от другой; каждая вновь добавляемая точка попадает в один из наибольших оставшихся интервалов и делит его в соответствии с золотым сечением! [Данное явление было впервые замечено Я. Одерфельдом (J. Oderfeld) и доказано С. Сверчковским (S. Świerczkowski), *Fundamenta Math.* **46** (1958), 187–189. В доказательстве важную роль играют числа Фибоначчи.]

Это замечательное свойство золотого сечения в действительности является частным случаем более общей теоремы, предложенной Хьюго Штейнгаузом (Hugo Steinhaus) и впервые доказанной Верой Туран Шош (Vera Turán Sós) [*Acta Math. Acad. Sci. Hung.* **8** (1957), 461–471; *Ann. Univ. Sci. Budapest. Eötvös Sect. Math.* **1** (1958), 127–134].

Теорема S. *Пусть θ — произвольное иррациональное число. При размещении точек $\{\theta\}$, $\{2\theta\}$, ..., $\{n\theta\}$ на отрезке $[0\,..\,1]$ длины $n+1$ образовавшихся отрезков имеют не более трех различных значений. Кроме того, очередная точка $\{(n+1)\theta\}$ попадет в один из наибольших уже полученных отрезков.* ▌

Таким образом, точки $\{\theta\}, \{2\theta\}, \ldots, \{n\theta\}$ расположены между 0 и 1 достаточно равномерно. Если θ — рациональное число, то теорема остается верна (при подходящей интерпретации отрезков нулевой длины, которые образуются при n, большем или равном знаменателю θ). Доказательство теоремы S вместе с подробным анализом сложившейся ситуации приводится в упр. 8. Оказывается, что отрезки данной длины создаются и уничтожаются в порядке очереди ("первым вошел — первым вышел"; first-in-first-out — FIFO). Конечно, одни значения θ лучше других; например, если θ близкó к 0 или 1, сначала получим много маленьких отрезков и один большой. В упр. 9 показано, что два числа, а именно — ϕ^{-1} и $\phi^{-2} = 1-\phi^{-1}$ приводят к "наиболее равномерно распределенным" последовательностям среди всех чисел θ между 0 и 1.

Рассмотренная теория подводит нас к *хешированию Фибоначчи*, при котором в качестве A берется ближайшее к $\phi^{-1}w$ целое число, взаимно простое с w. Напри-

мер, если бы MIX был десятичным компьютером, можно было бы воспользоваться множителем

$$A = \boxed{+}\boxed{61}\boxed{80}\boxed{33}\boxed{98}\boxed{87}\,. \tag{7}$$

Он хорошо рассеивает алфавитные ключи типа LIST1, LIST2, LIST3. Однако при рассмотрении изменения ключей в четвертой позиции — например, SUM1$_\sqcup$, SUM2$_\sqcup$, SUM3$_\sqcup$ — получим, что рассеяние происходит так же, как если бы теорема S использовалась с $\theta = \{100A/w\} = .80339887$ вместо $\theta = .6180339887 = A/w$. В результате данное рассеяние оказывается несколько хуже, чем при $\theta = \phi^{-1}$, хотя и остается достаточно хорошим. Однако в случае изменения во второй позиции ключа — A1$_{\sqcup\sqcup\sqcup}$, A2$_{\sqcup\sqcup\sqcup}$, A3$_{\sqcup\sqcup\sqcup}$ — эффективное значение θ равно .9887, что, пожалуй, слишком близко к 1.

Поэтому можно было бы работать с множителем

$$A = \boxed{+}\boxed{61}\boxed{61}\boxed{61}\boxed{61}\boxed{61}$$

вместо множителя (7); такой множитель будет хорошо разделять последовательности ключей, отличающиеся в *любой* позиции. К сожалению, подобный выбор приводит к новой проблеме, аналогичной возникающей при делении на $r^k \pm 1$: ключи типа XY и YX попадут при хешировании в одно и то же место! Один из путей обхода возникающих неприятностей начинается с более пристального рассмотрения структуры, лежащей в основе теоремы S. Для коротких последовательностей ключей важны лишь несколько первых частичных отношений представлений θ в виде цепной дроби. Маленькие частичные отношения соответствуют "хорошим" свойствам распределения. Поэтому можно показать, что наилучшие значения θ лежат в интервалах

$$\tfrac{1}{4} < \theta < \tfrac{3}{10}, \qquad \tfrac{1}{3} < \theta < \tfrac{3}{7}, \qquad \tfrac{4}{7} < \theta < \tfrac{2}{3}, \qquad \tfrac{7}{10} < \theta < \tfrac{3}{4}.$$

Значение A можно составить так, что каждый из его байтов будет находиться в "хорошем" интервале и будет не слишком близок к значениям других байтов (или их дополнениям), например

$$A = \boxed{+}\boxed{61}\boxed{25}\boxed{42}\boxed{33}\boxed{71}\,. \tag{8}$$

Такой множитель может быть смело рекомендован к использованию. (Изложенные в этом разделе идеи по мультипликативному хешированию в основном принадлежат Р. В. Флойду (R. W. Floyd).)

Хорошая хеш-функция должна удовлетворять двум требованиям.

a) Ее вычисление должно выполняться очень быстро.

b) Она должна минимизировать количество коллизий.

Свойство (a) зависит от особенностей компьютера, а свойство (b) — от особенностей данных. Если бы все ключи были действительно случайными, можно было бы использовать несколько битов из них и с их помощью получить хеш-функцию; однако на практике для удовлетворения (b) требуется функция, зависящая от всех битов ключа.

До сих пор мы рассматривали хеширование "однословных" ключей. С ключами, состоящими из нескольких слов, и с ключами переменной длины можно работать,

как с числами с многократной точностью и применять к ним все рассмотренные методы. Второй путь состоит в комбинировании слов в одно и в выполнении единственной операции умножения или деления, как описано выше. Эта комбинация может быть осуществлена путем сложения по модулю w или выполнения операции "исключающее или" бинарного компьютера. Обе операции используют все биты обоих аргументов; "исключающее или", однако, предпочтительнее, поскольку позволяет избежать арифметического переполнения. Основной недостаток такого метода заключается в том, что указанные операции коммутативны, а значит, хеш-адреса (X, Y) и (Y, X) будут совпадать. Чтобы устранить этот недостаток, Г. Д. Кнотт (G. D. Knott) предложил выполнять перед арифметической операцией циклический сдвиг.

Еще лучший путь хеширования состоящих из l символов (или l слов) ключей $K = x_1 x_2 \ldots x_l$ заключается в вычислении

$$h(K) = \big(h_1(x_1) + h_2(x_2) + \cdots + h_l(x_l)\big) \bmod M, \tag{9}$$

где каждое h_j представляет собой независимую хеш-функцию. Эта идея, предложенная Дж. Л. Картером (J. L. Carter) и М. Н. Вегманом (M. N. Wegman) в 1977 году, в особенности эффективна, когда каждый x_j представляет собой отдельный символ, поскольку в таком случае можно использовать предвычисленный массив для каждой функции h_j. Такие массивы делают умножение ненужным. И если M представляет собой степень двойки, то в (9) можно избежать деления, используя "исключающее или" вместо сложения. Таким образом, (9), определенно, удовлетворяет свойству (a). Более того, Картер и Вегман доказали, что если h_j выбирается случайным образом, то будет выполняться и свойство (b) *независимо от входных данных* (см. упр. 72).

Было предложено и множество других методов хеширования, однако ни для одного из них не было доказано его преимущество перед простыми методами, описанными выше. Обзор некоторых методов с детальными характеристиками их производительности при работе с реальными файлами можно найти в статье V. Y. Lum, P. S. T. Yuen, and M. Dodd, *CACM* **14** (1971), 228–239.

Из других интересных методов хеширования, видимо, наибольший интерес представляет технология, основанная на алгебраической теории кодирования. Эта технология аналогична методу деления, описанному выше, однако вместо деления на целое число осуществляется деление на полином по модулю 2. (Как упоминалось в разделе 4.6, эта операция аналогична делению, как сложение аналогично операции "исключающее или".) В данном методе M должно представлять степень 2 ($M = 2^m$), и мы используем полином m-й степени $P(x) = x^m + p_{m-1}x^{m-1} + \cdots + p_0$. n-цифровой бинарный ключ $K = (k_{n-1} \ldots k_1\, k_0)_2$ можно рассматривать как полином $K(x) = k_{n-1}x^{n-1} + \cdots + k_1 x + k_0$ и вычислить остаток

$$K(x) \bmod P(x) = h_{m-1}x^{m-1} + \cdots + h_1 x + h_0,$$

используя полиномиальную арифметику по модулю 2. В таком случае $h(K) = (h_{m-1} \ldots h_1\, h_0)_2$. При правильном выборе $P(x)$ эта хеш-функция может гарантировать отсутствие коллизий между почти одинаковыми ключами. Например, при $n = 15$, $m = 10$ и

$$P(x) = x^{10} + x^8 + x^5 + x^4 + x^2 + x + 1 \tag{10}$$

можно показать, что $h(K_1)$ будет всегда отлично от $h(K_2)$, если K_1 и K_2 — различные ключи, отличающиеся менее чем семью битами. (Дополнительная информация по этой теме приводится в упр. 7; в данном случае, конечно, более уместна аппаратная или микропрограммная реализация, а не программная.)

Очень часто удобно использовать постоянную хеш-функцию $h(K) = 0$ в процессе отладки программы, так как все ключи будут храниться вместе; эффективная хеш-функция $h(K)$ может быть подставлена позже.

Разрешение коллизий методом "цепочек". Мы уже говорили, что некоторые хеш-адреса могут быть одинаковыми для различных ключей. Вероятно, самый очевидный путь решения этой проблемы — поддержка M связных списков, по одному для каждого возможного значения хеш-кода. Поле LINK должно быть включено в каждую запись; кроме того, следует иметь M головных узлов списков, перенумерованных, скажем, от 1 до M. После хеширования ключа мы просто выполняем последовательный поиск в списке номер $h(K) + 1$. (Обратитесь к упр. 6.1–2. Ситуация весьма напоминает сортировку методом вставок в несколько списков; см. программу 5.2.1M.)

На рис. 38 показана простая схема с цепочками при $M = 9$ для последовательности из ключей

$$K = \texttt{EN, TO, TRE, FIRE, FEM, SEKS, SYV} \tag{11}$$

(представляющих числа от 1 до 7 на норвежском языке), имеющих хеш-коды

$$h(K) + 1 = 3,\ 1,\ 4,\ 1,\ 5,\ 9,\ 2 \tag{12}$$

соответственно. В первом списке содержится два элемента, три списка пусты.

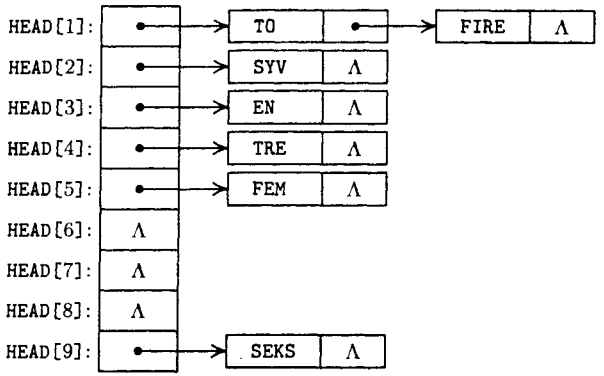


Рис. 38. Раздельные цепочки.

В связи с тем, что цепочки коротки, рассматриваемый здесь метод является весьма быстродействующим. Если 365 человек соберутся вместе в одной комнате, вероятно, окажется много пар, имеющих один и тот же день рождения, однако среднее количество людей с данным днем рождения равно только 1! В общем случае, если имеется N ключей и M списков, средний размер списка будет равен N/M; значит, хеширование приводит к уменьшению среднего количества работы, необходимой для последовательного поиска, примерно в M раз (точная формула выводится в упр. 34).

Этот метод представляет собой очевидную комбинацию обсуждавшихся ранее технологий, поэтому нет необходимости детально описывать алгоритм для хеш-таблиц с цепочками. Часто удобно хранить отдельные списки упорядоченными по ключам, что делает вставку и неудачный поиск более быстрыми. Так, если бы в приведенном примере использовались списки в порядке возрастания, то узлы TO и FIRE на рис. 38 поменялись бы местами, а все ссылки Λ были бы при этом заменены указателем на вспомогательную запись с ключом ∞ (см. алгоритм 6.1T). Другой подход предусматривает использование концепции "самоорганизованности", обсуждавшейся в разделе 6.1; записи могут быть упорядоченными не по значениям ключей списка, а по последнему времени обращения к ним.

Для повышения быстродействия желательны большие значения M, однако в этом случае слишком многие списки будут пусты и будут зря расходовать память на содержание заголовков списков. При небольших размерах записей можно воспользоваться следующим подходом: совместить пространство для записей с пространством для заголовков списков, отводя место для M записей и M ссылок вместо места для N записей и $M + N$ ссылок. Иногда можно совершать проход по всем данным для поиска заголовков списков, которые будут использоваться, а затем при втором проходе вставлять "переполняющие" записи в пустые "щели". Однако зачастую это непрактично или невозможно, поэтому хотелось бы иметь метод, работающий с каждой записью только раз — при ее помещении в систему. Следующий алгоритм, предложенный в работе F. A. Williams, *CACM* **2**, 6 (June, 1959), 21–24, позволяет быстро решить эту задачу.

Алгоритм C (*Поиск и вставка в хеш-таблице с цепочками*). Этот алгоритм (рис. 39) ищет данный ключ K в таблице из M узлов. Если K в таблице отсутствует, а таблица не заполнена, K вставляется в таблицу.

Узлы таблицы обозначаются через TABLE[i], $0 \le i \le M$, и могут быть двух различных типов — *пустыми* и *занятыми*. В занятом узле содержатся поле ключа KEY[i], поле ссылки LINK[i] и, возможно, другие поля.

Алгоритм использует хеш-функцию $h(K)$. Применяется также вспомогательная переменная R для упрощения поиска пустых мест; если таблица пуста, $R = M + 1$; по мере выполнения вставок всегда будет оставаться истинным утверждение, что узлы TABLE[j] заняты для всех j в диапазоне $R \le j \le M$. По соглашению узел TABLE[0] всегда пуст.

C1. [Хеширование.] Установить $i \leftarrow h(K) + 1$. (Теперь $1 \le i \le M$.)

C2. [Есть ли список?] Если TABLE[i] пуст, перейти к шагу C6. (В противном случае TABLE[i] занят и мы приступим к поиску в списке, начинающемся в этом узле.)

C3. [Сравнение.] Если $K =$ KEY[i], алгоритм успешно завершается.

C4. [Переход к следующему.] Если LINK[i] $\neq 0$, установить $i \leftarrow$ LINK[i] и вернуться к шагу C3.

C5. [Поиск пустого узла.] (Поиск был неудачным, и необходимо найти пустое место в таблице.) Уменьшать R один или более раз, пока не будет найдено такое значение, при котором узел TABLE[R] будет пуст. Если $R = 0$, алгоритм завершается в связи с переполнением (свободных узлов больше нет); в противном случае установить LINK[i] $\leftarrow R$, $i \leftarrow R$.

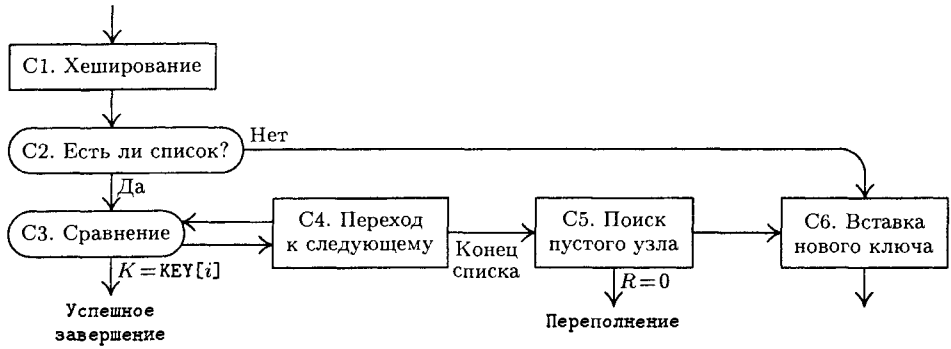


Рис. 39. Поиск и вставка в хеш-таблице с цепочками.

C6. [Вставка нового ключа.] Пометить узел TABLE[i] как занятый с KEY[i] $\leftarrow K$ и LINK[i] $\leftarrow 0$. ▌

Алгоритм позволяет объединить несколько списков, поэтому записи не нужно перемещать после вставки в таблицу. Например, взгляните на рис. 40, на котором SEKS попадает в список, содержащий TO и FIRE, поскольку последний уже был вставлен в позицию 9.

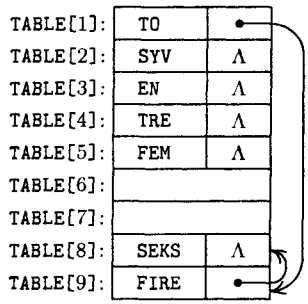


Рис. 40. "Сросшиеся" цепочки.

Для сравнения алгоритма C с другими алгоритмами этой главы можно написать следующую MIX-программу. Выполненный анализ показал, что цепочки, как правило, коротки, и приведенная программа написана с учетом этого факта.

Программа C (*Поиск и вставка в хеш-таблице с цепочками*). Для удобства считаем, что ключи состоят из трех байтов, а узлы имеют следующий вид:

Пустой узел	−	1	0	0	0	0	;
Занятый узел	+	LINK	KEY				.

(13)

В качестве размера таблицы M выбирается простое число; TABLE[i] хранится по адресу TABLE $+ i$. rI1 $\equiv i$, rA $\equiv K$.

```
01  KEY   EQU  3:5
02  LINK  EQU  0:2
```

03	START	LDX	K	1	*C1. Хеширование.*
04		ENTA	0	1	
05		DIV	=M=	1	
06		STX	*+1(0:2)	1	
07		ENT1	*	1	$i \leftarrow h(K)$
08		INC1	1	1	$+1$.
09		LDA	K	1	
10		LD2	TABLE,1(LINK)	1	*C2. Есть ли список?*
11		J2N	6F	1	Переход к C6, если TABLE[i] пуст.
12		CMPA	TABLE,1(KEY)	A	*C3. Сравнение.*
13		JE	SUCCESS	A	Выход, если $K =$ KEY[i].
14		J2Z	5F	$A - S1$	Переход к C5, если LINK[i] $= 0$.
15	4H	ENT1	0,2	$C - 1$	*C4. Переход к следующему.*
16		CMPA	TABLE,1(KEY)	$C - 1$	*C3. Сравнение.*
17		JE	SUCCESS	$C - 1$	Выход, если $K =$ KEY[i].
18		LD2	TABLE,1(LINK)	$C - 1 - S2$	
19		J2NZ	4B	$C - 1 - S2$	Продвижение при LINK[i] $\neq 0$.
20	5H	LD2	R	$A - S$	*C5. Поиск пустого узла.*
21		DEC2	1	T	$R \leftarrow R - 1$.
22		LDX	TABLE,2	T	
23		JXNN	*-2	T	Повторять, пока TABLE[R] пуст.
24		J2Z	OVERFLOW	$A - S$	Выход, если не осталось пустых узлов.
25		ST2	TABLE,1(LINK)	$A - S$	LINK[i] $\leftarrow R$.
26		ENT1	0,2	$A - S$	$i \leftarrow R$.
27		ST2	R	$A - S$	Обновление R в памяти.
28	6H	STZ	TABLE,1(LINK)	$1 - S$	*C6. Вставка нового ключа.* LINK[i] $\leftarrow 0$.
29		STA	TABLE,1(KEY)	$1 - S$	KEY[i] $\leftarrow K$. ▮

Время работы программы зависит от следующих параметров:

C — число проверенных во время поиска узлов таблицы;

A — [первая проверка обнаружила занятый узел];

S — [поиск был успешен];

T — количество элементов таблицы, проверенных при поиске пустого пространства.

Здесь $S = S1 + S2$, где $S1 = 1$ при успешной первой попытке. Общее время работы поисковой фазы программы C равно $(7C + 4A + 17 - 3S + 2S1)u$, а вставка нового ключа при $S = 0$ требует дополнительного времени $(8A + 4T + 4)u$.

Предположим, что в начале работы программы в таблице содержалось N ключей, и введем

$$\alpha = N/M = \text{коэффициент заполнения таблицы.} \tag{14}$$

Тогда среднее значение A при неудачном поиске, очевидно, равно α, если хеш-функция случайна; в упр. 39 доказывается, что среднее значение C для неудачного поиска равно

$$C'_N = 1 + \frac{1}{4}\left(\left(1 + \frac{2}{M}\right)^N - 1 - \frac{2N}{M}\right) \approx 1 + \frac{e^{2\alpha} - 1 - 2\alpha}{4}. \tag{15}$$

Следовательно, если таблица заполнена наполовину, среднее число проб, выполняемых при неудачном поиске, составляет около $\frac{1}{4}(e+2) \approx 1.18$. И даже если таблица заполнена полностью, среднее количество проб при вставке последнего элемента равно всего лишь $\frac{1}{4}(e^2+1) \approx 2.10$. Стандартное отклонение также невелико (что показано в упр. 40). Приведенные статистические выкладки показывают, что при случайной хеш-функции *списки остаются короткими, хотя алгоритм допускает их "срастание"*. Конечно, C может оказаться равным даже N, если функция плоха или вы — исключительно невезучий человек...

При успешном завершении поиска A всегда равно 1. Среднее количество проб можно вычислить путем суммирования величин $C+A$ по первым N неудачным поискам и деления на N в предположении, что все ключи равновероятны. Таким образом, получаем следующее среднее количество проб при случайном успешном поиске:

$$C_N = \frac{1}{N}\sum_{0\le k<N}\left(C'_k + \frac{k}{M}\right) = 1 + \frac{1}{8}\frac{M}{N}\left(\left(1+\frac{2}{M}\right)^N - 1 - \frac{2N}{M}\right) + \frac{1}{4}\frac{N-1}{M}$$
$$\approx 1 + \frac{e^{2\alpha}-1-2\alpha}{8\alpha} + \frac{\alpha}{4}. \tag{16}$$

Даже если таблица будет полностью заполненной, в среднем потребуется только 1.80 пробы для нахождения элемента! Аналогично (см. упр. 42) среднее значение $S1$ равно

$$S1_N = 1 - \tfrac{1}{2}\big((N-1)/M\big) \approx 1 - \tfrac{1}{2}\alpha. \tag{17}$$

На первый взгляд может показаться, что шаг C5 неэффективен, поскольку поиск пустой позиции на нем осуществляется последовательно. Но в действительности общее количество проб на шаге C5 при построении таблицы никогда не будет превышать количества элементов в таблице; значит, в среднем мы делаем не более одной такой пробы на вставку. В упр. 41 доказывается, что при случайном неудачном поиске $T \approx \alpha e^{\alpha}$.

Можно модифицировать алгоритм C так, что никакие два списка не будут "срастаться", но тогда придется прибегнуть к перемещению записей. Например, рассмотрим ситуацию, представленную на рис. 40, непосредственно перед вставкой SEKS в позицию 9. Чтобы списки оставались раздельными, необходимо переместить FIRE, а для этого нужно найти узел, указывающий на FIRE. Решить эту проблему можно, используя не двусвязный список, а хеширование FIRE и поиск в его списке, как было предложено Д. Э. Фергюсоном (D. E. Ferguson), поскольку списки невелики по размеру. В упр. 34 показано, что среднее количество проб в случае раздельных списков уменьшается до

$$C'_N = 1 + \frac{N(N-1)}{2M^2} \approx 1 + \frac{\alpha^2}{2} \quad \text{(неудачный поиск)}; \tag{18}$$

$$C_N = 1 + \frac{N-1}{2} \approx 1 + \frac{\alpha}{2} \quad \text{(успешный поиск)}. \tag{19}$$

Это, пожалуй, не слишком большое улучшение по сравнению с (15) и (16), чтобы прибегать к такому изменению алгоритма.

С другой стороны, Батлер Лэмпсон (Butler Lampson) заметил, что бóльшая часть пространства, в действительности занятая ссылками, может быть сэкономлена при использовании метода цепочек, если избавиться от "срастания" списков. Это позволяет получить интересный алгоритм, обсуждающийся в упр. 13. Метод Лэмпсона вводит дополнительный бит в каждый элемент, слегка уменьшая при этом среднее количество проб, которые необходимы при неудачном поиске: с (18) до

$$\left(1-\frac{1}{M}\right)^N+\frac{N}{M}\approx e^{-\alpha}+\alpha. \tag{18'}$$

Раздельные цепочки, показанные на рис. 38, могут использоваться при $N > M$; и, таким образом, проблема переполнения снимается. При срастании списков (см. рис. 40 и алгоритм C) можно поместить переполняющие элементы во вспомогательный пул памяти; Л. Гюиба (L. Guibas) доказал, что среднее количество проб при вставке $(M+L+1)$-го элемента составляет $\left(L/2M+\frac{1}{4}\right)\left((1+2/M)^M-1\right)+\frac{1}{2}$. Однако обычно предпочтительнее использовать альтернативную схему, при которой первые вызывающие коллизию элементы помещаются во вспомогательную область памяти; в таком случае списки будут "срастаться" только при заполнении вспомогательной области (см. упр. 43).

Разрешение коллизий методом открытой адресации. Другой путь решения проблемы, связанной с коллизиями, состоит в том, чтобы полностью отказаться от ссылок, просто просматривая различные записи таблицы одну за одной до тех пор, пока не будет найден ключ K или пустая позиция. Идея заключается в формулировании правила, согласно которому по данному ключу K определяется "пробная последовательность", т. е. последовательность позиций таблицы, которые должны быть просмотрены при вставке или поиске K. Если при поиске K согласно определенной этим ключом последовательности встречается пустая позиция, можно сделать вывод о том, что K в таблице отсутствует (поскольку та же последовательность проверок должна выполняться при каждом поиске K). Этот общий класс методов назван *открытой адресацией* [см. W. W. Peterson, *IBM J. Research & Development* **1** (1957), 130–146].

Простейшая схема открытой адресации, известная как *линейное исследование*, использует циклическую последовательность проверок

$$h(K),\ h(K)-1,\ \ldots,\ 0,\ M-1,\ M-2,\ \ldots,\ h(K)+1 \tag{20}$$

и описывается следующим образом.

Алгоритм L (*Линейное исследование и вставка*). Этот алгоритм выполняет поиск данного ключа K в таблице с M узлами. Если K отсутствует в таблице и таблица не полна, ключ K будет вставлен в таблицу.

Узлы таблицы обозначаются как `TABLE[`i`]`, $0 \le i < M$, и могут быть двух типов — *пустыми* и *занятыми*. В занятых узлах содержатся ключи `KEY[`i`]` и, возможно, другие поля. Вспомогательная переменная N используется для отслеживания количества занятых узлов; она рассматривается как часть таблицы и увеличивается на 1 при каждой вставке нового ключа.

Алгоритм использует хеш-функцию $h(K)$ и линейную последовательность проб (20) для адресации таблицы. Модификации этой последовательности обсуждаются ниже.

L1. [Хеширование.] Установить $i \leftarrow h(K)$. (Теперь $0 \le i < M$.)

L2. [Сравнение.] Если узел TABLE[i] пуст, перейти к шагу L4. В противном случае, если KEY[i] $= K$, алгоритм успешно завершается.

L3. [Переход к следующему.] Установить $i \leftarrow i - 1$; если теперь $i < 0$, установить $i \leftarrow i + M$. Вернуться к шагу L2.

L4. [Вставка.] (Поиск оказался неудачным.) Если $N = M - 1$, алгоритм завершается в связи с переполнением (условие полного заполнения таблицы полностью в данном алгоритме выглядит как $N = M - 1$, а не как $N = M$; см. упр. 15). В противном случае установить $N \leftarrow N + 1$, пометить узел TABLE[i] как занятый и установить KEY[i] $\leftarrow K$. ▮

На рис. 41 показано, что происходит при вставке семи ключей нашего примера с норвежскими цифрами согласно алгоритму L из примера с хеш-кодами 2, 7, 1, 8, 2, 8, 1; при этом последние три ключа, FEM, SEKS и SYV, смещены со своих начальных позиций $h(K)$.

0	FEM
1	TRE
2	EN
3	
4	
5	SYV
6	SEKS
7	TO
8	FIRE

Рис. 41. Линейная открытая адресация.

Программа L (*Линейное исследование и вставка*). Эта программа работает с ключами, занимающими полное слово; однако ключ 0 использовать нельзя, поскольку нулевое значение ключа означает, что данный узел в таблице свободен (другое решение состоит, например, в том, чтобы наложить на ключи условие неотрицательности, а пустые позиции пометить значением -1). Размер таблицы — M (предполагается, что это простое число), а узлы таблицы TABLE[i] имеют адреса TABLE$+i$, $0 \le i < M$. Для ускорения внутреннего цикла в ячейке TABLE $-$ 1 содержится 0. В ячейке VACANCIES хранится значение $M - 1 - N$; и, наконец, rA $\equiv K$, rI1 $\equiv i$.

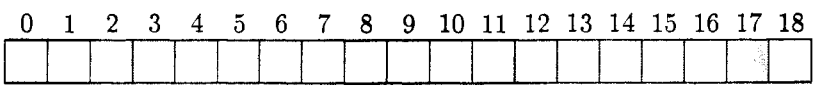
Для ускорения внутреннего цикла этой программы из него удалена проверка "$i < 0$", так что в нем остались только существенные части шагов L2 и L3. Общее время работы программы на фазе поиска составляет $(7C + 9E + 21 - 4S)u$, а вставка в случае неудачного поиска добавляет к этой величине еще $8u$.

```
01  START LDX  K            1      L1. Хеширование.
02        ENTA 0            1
03        DIV  =M=          1
04        STX  *+1(0:2)     1
05        ENT1 *            1      i ← h(K).
06        LDA  K            1
07        JMP  2F           1
```

08	8H	INC1	M+1	E	*L3. Переход к следующему.*
09	3H	DEC1	1	$C+E-1$	$i \leftarrow i-1$.
10	2H	CMPA	TABLE,1	$C+E$	*L2. Сравнение.*
11		JE	SUCCESS	$C+E$	Выход, если $K =$ KEY[i].
12		LDX	TABLE,1	$C+E-S$	
13		JXNZ	3B	$C+E-S$	Переход к шагу L3 при непустом TABLE[i].
14		J1N	8B	$E+1-S$	Переход к шагу L3 с $i \leftarrow M$, если $i=-1$.
15	4H	LDX	VACANCIES	$1-S$	*L4. Вставка.*
16		JXZ	OVERFLOW	$1-S$	Выход по условию переполнения, если $N = M-1$.
17		DECX	1	$1-S$	
18		STX	VACANCIES	$1-S$	Увеличение N на 1.
19		STA	TABLE,1	$1-S$	TABLE[i] $\leftarrow K$. ▌

Как и в программе C, переменная C описывает количество проб, а S указывает, успешным ли был поиск. Проигнорировать переменную E, которая равна 1, можно только в случае проверки фиктивного узла TABLE[−1]; ее среднее значение равно $(C-1)/M$.

Опыты с линейным исследованием показывают, что алгоритм хорошо работает в начале заполнения таблицы, однако по мере заполнения процесс замедляется, а длинные серии проб становятся все более частыми. Причину такого поведения можно понять, рассмотрев следующую гипотетическую хеш-таблицу при $M = 19$ и $N = 9$.



(21)

Темные квадраты представляют собой занятые позиции. Следующий ключ K, вставляемый в таблицу, попадет в одно из десяти пустых мест, которые, однако, отнюдь не равновероятны. Так, K будет вставлен в позицию 11 при $11 \le h(K) \le 15$, в то время как в позицию 8 он попадет, только когда $h(K) = 8$. Следовательно, вероятность попадания в ячейку 11 в пять раз больше, чем вероятность попадания в ячейку 8; общая тенденция такова, что длинные списки стремятся стать еще длиннее.

Этого явления, тем не менее, самого по себе недостаточно для описания происходящего, поскольку подобная ситуация складывается и при использовании алгоритма C (вероятность увеличения списка из четырех элементов в четыре раза больше вероятности увеличения списка из одного элемента). Впрочем, настоящие неприятности начинаются при вставке в ячейку типа 4 или 16, когда разные списки объединятся в один (в то время как в алгоритме C списки никогда не удлинялись при вставке более чем на 1 элемент). Следовательно, при приближении N к M производительность алгоритма резко снижается.

Ниже в этом разделе будет доказано, что среднее количество проб, требуемое алгоритмом L, составляет примерно

$$C'_N \approx \frac{1}{2}\left(1+\left(\frac{1}{1-\alpha}\right)^2\right) \quad \text{(неудачный поиск)}, \tag{22}$$

$$C_N \approx \frac{1}{2}\left(1+\frac{1}{1-\alpha}\right) \quad \text{(успешный поиск)}, \tag{23}$$

где $\alpha = N/M$ — коэффициент заполненности таблицы. Таким образом, программа L практически столь же быстра, как и программа C при заполненности таблицы менее чем на 75%, несмотря на то что программа C работает с неправдоподобно короткими ключами. С другой стороны, когда $\alpha \to 1$, лучшее, что можно сказать о программе L, — это то, что она работает медленно, но верно. В самом деле, при $N = M - 1$ в таблице имеется только одно вакантное место, а потому среднее число проверок при неудачном поиске составит $(M + 1)/2$; мы докажем также, что среднее количество проб при успешном поиске в заполненной таблице равно примерно $\sqrt{\pi M/8}$.

Явление скучивания, которое делает линейное исследование дорогостоящим при почти заполненных таблицах, усугубляется при хешировании методом деления, если вероятно появление последовательных значений ключей $\{K, K+1, K+2, \ldots\}$, поскольку эти ключи будут иметь последовательные хеш-коды. Мультипликативное хеширование позволяет успешно разбить такие кластеры.

Другой путь защиты от последовательных хеш-кодов состоит в установке на шаге L3 $i \leftarrow i - c$ вместо $i \leftarrow i - 1$. Можно использовать любое положительное значение c, *взаимно простое* с M, поскольку последовательность проб в этом случае будет проверять в конечном счете все позиции таблицы без исключения. Такое изменение немного замедлит работу программы L из-за проверки $i < 0$. Уменьшение на c вместо уменьшения на 1 не устранит явление скучивания, так как теперь будут формироваться группы записей на расстоянии c одна от другой; формулы (22) и (23) останутся приемлемыми. Однако в этой ситуации последовательные ключи $\{K, K+1, K+2, \ldots\}$ станут не помехой, а помощниками.

Хотя фиксированное значение c не приводит к устранению скучивания, можно существенно улучшить ситуацию, сделав c зависящим от K. Эта идея позволяет выполнить важную модификацию алгоритма L, впервые предложенного Гюи де Бальбином (Guy de Balbine) [Ph. D. thesis, Calif. Inst. of Technology (1968), 149–150].

Алгоритм D (*Открытая адресация с двойным хешированием*). Этот алгоритм почти идентичен алгоритму L, но проверяет таблицу немного иначе, используя две хеш-функции: $h_1(K)$ и $h_2(K)$. Как обычно, значения $h_1(K)$ находятся в диапазоне от 0 до $M - 1$ включительно; функция $h_2(K)$ должна порождать значения от 1 до $M - 1$, *взаимно простые* с M. (Например, если M простое число, то $h_2(K)$ может быть любой величиной от 1 до $M - 1$ включительно; или, если $M = 2^m$, $h_2(K)$ может быть любым *нечетным* числом от 1 до $2^m - 1$.)

D1. [Первое хеширование.] Установить $i \leftarrow h_1(K)$.

D2. [Первая проба.] Если TABLE[i] пуст, перейти к шагу D6. В противном случае, если KEY[i] $= K$, алгоритм завершается успешно.

D3. [Второе хеширование.] Установить $c \leftarrow h_2(K)$.

D4. [Переход к следующему.] Установить $i \leftarrow i - c$; если после этого $i < 0$, установить $i \leftarrow i + M$.

D5. [Сравнение.] Если TABLE[i] пуст, перейти к шагу D6. В противном случае, если KEY[i] $= K$, алгоритм завершается успешно; иначе — вернуться к шагу D4.

D6. [Вставка.] Если $N = M - 1$, выполнение алгоритма прекращается в связи с переполнением. В противном случае установить $N \leftarrow N + 1$, пометить узел TABLE[i] как занятый и установить KEY[i] $\leftarrow K$. ▌

Для вычисления $h_2(K)$ было предложено несколько различных вариантов. Если M — простое число и $h_1(K) = K \bmod M$, можно положить $h_2(K) = 1 + (K \bmod (M-1))$; однако, поскольку $M-1$ четно, было бы лучше положить $h_2(K) = 1 + (K \bmod (M-2))$. Это приводит к мысли о таком выборе M, что M и $M-2$ были "простыми близнецами" наподобие 1021 и 1019. Кроме того, можно взять $h_2(K) = 1 + (\lfloor K/M \rfloor \bmod (M-2))$ в связи с тем, что частное $\lfloor K/M \rfloor$ может быть получено в регистре как побочный результат вычисления $h_1(K)$.

Если $M = 2^m$ и используется мультипликативное хеширование, $h_2(K)$ может быть вычислена методом простого сдвига на m дополнительных битов влево с выполнением операции "ИЛИ" с 1, так что к последовательности команд (5) необходимо добавить следующее.

```
ENTA 0     Очистить rA.
SLB  m     Сдвинуть rAX на m бит влево.          (24)
OR   =1=   rA ← rA ∨ 1.
```

Эти операции выполняются быстрее метода деления.

В каждом из предложенных выше методов $h_1(K)$ и $h_2(K)$ существенно независимы — в том смысле, что для различных ключей вероятность совпадения h_1 и h_2 пропорциональна $1/M^2$, а не $1/M$. Эмпирические тесты показывают, что число проб в алгоритме D с независимыми хеш-функциями неотличимо от числа проб, требующихся при случайной вставке ключей в таблицу; в этой ситуации практически отсутствует "скучивание", или "кластеризация", как в алгоритме L.

Можно также допустить зависимость $h_2(K)$ от $h_1(K)$, как было предложено Гари Кноттом (Gary Knott) в 1968 году; например, при простом M можно положить

$$h_2(K) = \begin{cases} 1, & \text{если } h_1(K) = 0; \\ M - h_1(K), & \text{если } h_1(K) > 0. \end{cases} \tag{25}$$

Этот метод выполняется быстрее повторного деления, однако он вызывает определенную *вторичную кластеризацию*, что приводит к небольшому увеличению числа проб из-за повышения вероятности того, что два или несколько ключей пойдут по одному и тому же пути. Выведенные ниже формулы могут использоваться для определения того, превышает ли выигрыш во времени хеширования потери времени на дополнительных пробах.

Алгоритмы L и D очень похожи, однако между ними найдется достаточно различий, чтобы было полезно сравнить время работы соответствующих MIX-программ.

Программа D (*Открытая адресация с двойным хешированием*). Поскольку эта программа очень похожа на программу L, она представлена здесь без комментариев. В программе rI2 $\equiv c - 1$.

```
01 START LDX  K          1          10     SRAX 5          A − S1
02       ENTA 0          1          11     DIV  =M-2=      A − S1
03       DIV  =M=        1          12     STX  *+1(0:2)   A − S1
04       STX  *+1(0:2)   1          13     ENT2 *          A − S1
05       ENT1 *          1          14     LDA  K          A − S1
06       LDX  TABLE,1    1          15  3H DEC1 1,2        C − 1
07       CMPX K          1          16     J1NN *+2        C − 1
08       JE   SUCCESS    1          17     INC1 M            B
09       JXZ  4F       1 − S1       18     CMPA TABLE,1    C − 1
```

```
19      JE   SUCCESS       C − 1          24  DECX 1            1 − S
20      LDX  TABLE,1     C − 1 − S2       25  STX  VACANCIES    1 − S
21      JXNZ 3B          C − 1 − S2       26  LDA  K            1 − S
22   4H LDX  VACANCIES     1 − S          27  STA  TABLE,1      1 − S
23      JXZ  OVERFLOW      1 − S
```

Счетчики частот A, C, $S1$, $S2$ здесь имеют тот же смысл, что и в программе C, приведенной выше. Еще одна переменная, B, в среднем примерно равна $(C-1)/2$ (если ограничить диапазон значений $h_2(K)$ до, скажем, $1 \le h_2(K) \le M/2$, B уменьшится до $(C-1)/4$; такое увеличение скорости, вероятно, *не* компенсируется заметным увеличением числа проб). Если в таблице $N = \alpha M$ ключей, среднее значение A, конечно, равно α при неудачном поиске и $A = 1$ — в случае поиска успешного. Как и в алгоритме C, среднее значение $S1$ при успешном поиске составляет $1 - \frac{1}{2}((N-1)/M) \approx 1 - \frac{1}{2}\alpha$. Среднее число проб трудно установить точно, однако эмпирические тесты показывают хорошую согласованность с формулами, выведенными ниже для "равномерного исследования" при независимых $h_1(K)$ и $h_2(K)$:

$$C'_N = \frac{M+1}{M+1-N} \approx (1-\alpha)^{-1} \quad \text{(неудачный поиск)}, \tag{26}$$

$$C_N = \frac{M+1}{N}(H_{M+1} - H_{M+1-N}) \approx -\alpha^{-1}\ln(1-\alpha) \quad \text{(успешный поиск)}. \tag{27}$$

Если $h_2(K)$ зависит от $h_1(K)$ согласно (25), вторичная кластеризация увеличивает (26) и (27) до

$$C'_N = \frac{M+1}{M+1-n} - \frac{N}{M+1} + H_{M+1} - H_{M+1-N} + O(M^{-1}) \approx (1-\alpha)^{-1} - \alpha - \ln(1-\alpha); \tag{28}$$

$$C_N = 1 + H_{M+1} - H_{M+1-N} - \frac{N}{2(M+1)} - (H_{M+1} - H_{M+1-N})/N + O(N^{-1}) \approx 1 - \ln(1-\alpha) - \tfrac{1}{2}\alpha \tag{29}$$

(см. упр. 44). Заметим, что при заполнении таблицы данные значения C_N стремятся к $H_{M+1} - 1$ и $H_{M+1} - \frac{1}{2}$ соответственно; это существенно лучше, чем при использовании алгоритма L, но не так хорошо, как в методе цепочек.

Поскольку каждая проба в алгоритме L отнимает немного меньше времени, двойное хеширование дает выигрыш только при заполненной таблице. На рис. 42 сравниваются средние значения времени работы программ L, D и модифицированной программы D (эта модификация влечет за собой вторичное скучивание; сама модификация сводится к замещению медленного вычисления $h_2(K)$ в строках 10–13 следующими командами.

```
ENN2 1-M,1    c ← M − i.
J1NZ *+2                                   (30)
ENT2 0        Если i = 0, c ← 1.
```

Программа D требует в целом $8C + 19A + B + 26 - 13S - 17S1$ единиц времени; модификация (30) позволяет сохранить около $15(A - S1) \approx 7.5\alpha$ времени при успешном завершении поиска. В данном случае вторичная кластеризация предпочтительнее независимого двойного хеширования.

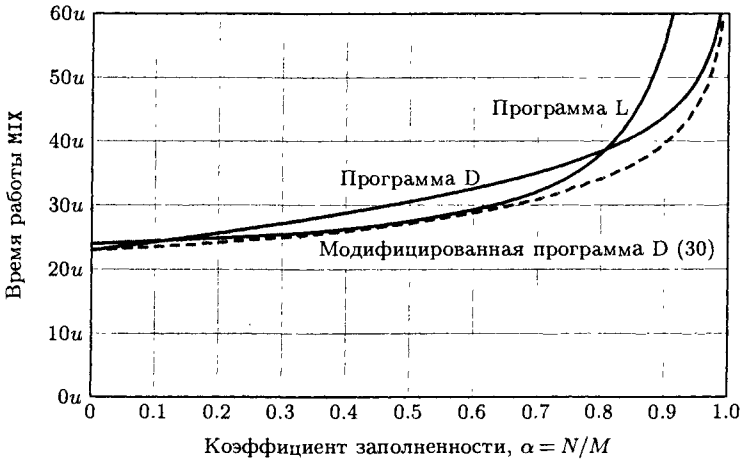


Рис. 42. Время успешного поиска трех схем открытой адресации.

На двоичном компьютере можно было бы увеличить скорость вычисления $h_2(K)$ другим путем — заменив (если M — простое число, большее, чем, скажем, 512) строки 10–13 строками

```
AND  =511=      rA ← rA mod 512.
STA  *+1(0:2)                                   (31)
ENT2 *          c ← rA + 1.
```

Эта идея (предложенная Беллом (Bell) и Каманом (Kaman), *CACM* **13** (1970), 675–677, которые независимо открыли алгоритм D) позволяет избежать вторичной кластеризации без затрат на повторное деление.

Для улучшения алгоритма L было предложено много других последовательностей проб, но ни одна из них не превосходит алгоритм D (за исключением, возможно, метода, описанного в упр. 20).

Используя относительный порядок ключей, можно уменьшить среднее время работы при неудачном поиске согласно алгоритму L или D до среднего времени работы при успешном поиске (см. упр. 66). Эта технология может с успехом применяться для решения задач, в которых неудачный поиск — обычное явление, более частое, чем поиск успешный; например, TEX использует такой алгоритм при поиске исключений из своих правил переноса.

Изменение Брента. Ричард П. Брент (Richard P. Brent) открыл такой способ модификации алгоритма D, при котором среднее время успешного поиска остается ограниченным при заполнении таблицы. Его метод [*CACM* **16** (1973), 105–109] основан на том факте, что обычно успешный поиск встречается гораздо чаще вставок, а потому его предложение сводится к переносу основной работы на вставку элемента путем такого перемещения записей, чтобы уменьшилось ожидаемое время выборки.

Например, на рис. 43 показано, сколько раз каждый идентификатор встречался в типичной процедуре PL/I. Эти данные указывают на то, что компилятор PL/I, который использует хеш-таблицу для хранения имен переменных, будет обращаться к ней за многими именами пять и более раз, вставляя их всего однажды. В подобном исследовании Белл и Каман обнаружили, что компилятор COBOL использовал свой

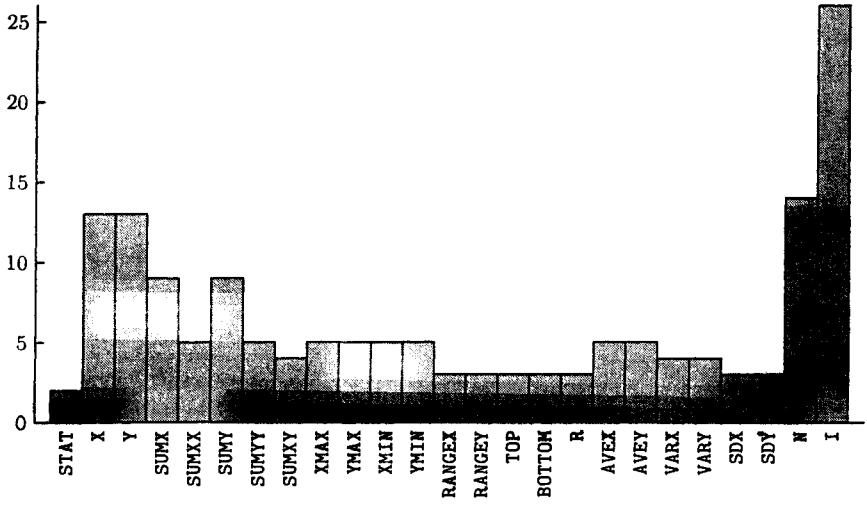


Рис. 43. Количество поисков имен переменных компилятором в типичном случае. Имена перечислены слева направо в порядке их первого появления в программе.

табличный алгоритм при компиляции 10 988 раз, в то время как было сделано только 735 вставок в таблицу; следовательно, на одну вставку приходилось около 14 успешных поисков. Иногда таблица создается только один раз (например, таблица символов кодов операций в ассемблере) и используется исключительно для получения из нее данных.

Идея Брента заключается в изменении процесса вставки в алгоритме D, как показано ниже. Предположим, что при неудачном поиске были проверены ячейки $p_0, p_1, \ldots, p_{t-1}, p_t$, где $p_j = \big(h_1(K) - jh_2(K)\big) \bmod M$ и TABLE[p_t] пуст. Если $t \le 1$, вставляем K в позицию p_t, как обычно; однако, если $t \ge 2$, вычисляем $c_0 = h_2(K_0)$, где $K_0 =$ KEY[p_0], и смотрим, свободен ли узел TABLE[$(p_0 - c_0) \bmod M$]. Если это так, помещаем в него TABLE[p_0], а затем вставляем K в позицию p_0. Это приводит к увеличению времени выборки K_0 на один шаг, но время выборки K уменьшается на $t \ge 2$ шагов, что приводит к общему выигрышу во времени выборки. Аналогично, если узел TABLE[$(p_0 - c_0) \bmod M$] занят и $t \ge 3$, проверяем узел TABLE[$(p_0 - 2c_0) \bmod M$]; если он также занят, вычисляем $c_1 = h_2$(KEY[p_1]), проверяем TABLE[$(p_0 - c_1) \bmod M$] и т. д. Вообще, пусть $c_j = h_2$(KEY[p_j]) и $p_{j,k} = (p_j - kc_j) \bmod M$; если для всех j и k, таких, что $j + k < r$, узлы TABLE[$p_{j,k}$] заняты и если $t \ge r + 1$, просматриваем узлы TABLE[$p_{0,r}$], TABLE[$p_{1,r-1}$], ..., TABLE[$p_{r-1,1}$]. Если первое пустое место оказывается в позиции $p_{j,r-j}$, устанавливаем TABLE[$p_{j,r-j}$] $\leftarrow$ TABLE[p_j] и вставляем K в позицию p_j.

Анализ Брента показывает, что среднее количество проб на один успешный поиск уменьшается до уровней, показанных на рис. 44 на с. 583, с максимальным значением, равным приблизительно 2.49.

Количество проб $t + 1$ при неудачном поиске в результате внесенных в алгоритм изменений осталось прежним; оно осталось на уровне, который определен формулой (26) и достигает $\frac{1}{2}(M + 1)$ при заполненности таблицы. Среднее количество

вычислений функции h_2 для одной вставки составляет согласно анализу Брента $\alpha^2+\alpha^5+\frac{1}{3}\alpha^6+\cdots$ и достигает в конечном счете $\Theta(\sqrt{M})$; количество дополнительных проб позиций таблицы после решения о том, каким образом будет производиться вставка, составляет около $\alpha^2+\alpha^4+\frac{4}{3}\alpha^5+\alpha^6+\cdots$.

С усовершенствованными модификациями Брента работал Е. Г. Маллах (E. G. Mallach) [*Comp. J.* **20** (1977), 137–140]; с другими результатами в этом направлении можно ознакомиться в работе Gaston H. Gonnet and J. Ian Munro, *SICOMP* **8** (1979), 463–478.

Удаления. Многие программисты настолько слепо верят в алгоритмы, что даже не пытаются задуматься о том, как они работают. Для них неприятным сюрпризом становится то, что *очевидный способ удаления записей из хеш-таблицы не работает*. Например, чтобы удалить ключ EN на рис. 41, нельзя просто пометить этот узел в таблице как пустой, поскольку окажется потерянным другой ключ, FEM. (Вспомним, что и EN, и FEM хешируются в одно и то же положение. При поиске FEM мы натолкнемся на пустое место, что свидетельствует о неудачном завершении поиска.) Подобная проблема возникает в случае использования алгоритма C из-за срастания списков; рассмотрите удаление двух ключей — TO и FIRE — на рис. 40.

Вообще говоря, обрабатывать удаление можно, поместив в соответствующую ячейку специальный код. Таким образом, будет иметься три вида позиций в таблице: пустые, занятые и удаленные. При поиске ключа следует пропускать удаленные ячейки, как если бы они были заняты. В случае неудачного поиска ключ может быть помещен в первую встреченную удаленную или пустую позицию.

Однако такой метод работает только при редких удалениях, поскольку однажды занятая позиция больше никогда не сможет стать свободной, а значит, после длинной последовательности вставок и удалений все свободные позиции исчезнут, а при неудачном поиске будет требоваться M проверок! Более того, время пробы при этом увеличится, так как на шаге D4 придется проверять, не приняло ли i свое первоначальное значение, а число проб в случае успешного поиска возрастет с C_N до C'_N.

При использовании линейного исследования (алгоритм L) можно поступить более разумно, если, конечно, затратить дополнительные усилия на удаление.

Алгоритм R (*Удаление при линейном исследовании*). Пусть хеш-таблица построена по алгоритму L. Алгоритм удаляет запись из данной позиции TABLE[i].

R1. [Освобождение ячейки.] Пометить TABLE[i] как пустую и установить $j \leftarrow i$.

R2. [Уменьшение i.] Установить $i \leftarrow i-1$ и, если i стало отрицательным, установить $i \leftarrow i+M$.

R3. [Проверить TABLE[i].] Если TABLE[i] пуст, алгоритм завершается. В противном случае установить $r \leftarrow h(\text{KEY}[i])$ — первоначальный хеш-адрес ключа, который хранится в позиции i. Если $i \le r < j$ или если $r < j < i$ либо $j < i \le r$ (другими словами, если r лежит циклически между i и j), вернуться к шагу R1.

R4. [Переместить запись.] Установить TABLE[j] $\leftarrow$ TABLE[i] и вернуться к шагу R1. ∎

В упр. 22 показано, что этот алгоритм не вызывает снижения производительности; другими словами, среднее количество проб, предсказанное в формулах (22) и (23), останется тем же (более слабый результат для вставки в дерево был доказан

в теореме 6.2.2Н). Однако корректность алгоритма R сильно зависит от того факта, что используется линейное исследование хеш-таблицы, а потому аналогичный алгоритм удаления при применении алгоритма D отсутствует. Среднее время работы алгоритма R анализируется в упр. 64.

Конечно, при использовании метода цепочек с различными списками для всех возможных хеш-значений удаление не вызывает проблем, поскольку сводится к простому удалению из связанного линейного списка. Удаление для алгоритма C обсуждается в упр. 23.

Алгоритм R позволяет перемещать некоторые элементы таблицы, что может оказаться нежелательно (например, если на элементы имеются ссылки извне). Другой подход к проблеме удаления основывается на адаптировании некоторых идей, использующихся при сборке мусора (см. раздел 2.3.5): можно хранить количество ссылок с каждым ключом, говорящим о том, как много других ключей сталкивается с ним; тогда при обнулении счетчика можно преобразовать такие ячейки в пустые. Другой метод состоит в проходе по всей таблице, как только наберется слишком много удаленных элементов, замене всех незанятых позиций пустыми и просмотре всех оставшихся ключей, чтобы выяснить, какие незанятые позиции все еще имеют статус "удаленных". Эти методы, позволяющие избежать перемещения элементов в таблице и работающие с любой хеш-технологией, были впервые предложены в работе T. Gunji and E. Goto, *J. Information Proc.* **3** (1980), 1–12.

***Анализ алгоритмов.** Поскольку при хешировании необходимо опираться на теорию вероятностей, очень важно знать поведение метода хеширования в среднем. Наихудший случай использования этих алгоритмов невообразимо плох, поэтому следует убедиться в том, что поведение в среднем *очень* хорошее.

Прежде чем приступить к анализу линейного исследования и других методов хеширования, рассмотрим приближенную модель ситуации, называемой *равномерным исследованием*. В этой модели, предложенной В. В. Петерсоном (W. W. Peterson) [*IBM J. Research & Development* **1** (1957), 135–136], предполагается, что каждый ключ размещается в совершенно случайном месте таблицы, так что все $\binom{M}{N}$ возможных конфигураций из N занятых и $M-N$ пустых ячеек равновероятны. В данной модели игнорируются эффекты первичного или вторичного скучивания; предполагается, что занятость каждой конкретной ячейки не зависит от занятости остальных. Тогда вероятность того, что для вставки $(N+1)$-го элемента необходимо в точности r проб, равна количеству конфигураций, в которых $r-1$ данная ячейка занята, а еще одна пуста, деленному на $\binom{M}{N}$, т. е.

$$P_r = \binom{M-r}{N-r+1} \Big/ \binom{M}{N}.$$

Таким образом, среднее количество проб в случае равномерного исследования составляет

$$\begin{aligned} C'_N &= \sum_{r=1}^{M} rP_r = M+1-\sum_{r=1}^{M}(M+1-r)P_r \\ &= M+1-\sum_{r=1}^{M}(M+1-r)\binom{M-r}{M-N-1}\Big/\binom{M}{N} \end{aligned}$$

$$= M+1-\sum_{r=1}^{M}(M-N)\binom{M+1-r}{M-N}\Big/\binom{M}{N}$$
$$= M+1-(M-N)\binom{M+1}{M-N+1}\Big/\binom{M}{N}$$
$$= M+1-(M-N)\frac{M+1}{M-N+1} = \frac{M+1}{M-N+1}, \qquad 1 \le N < M. \tag{32}$$

(Мы уже решали, по существу, ту же задачу в связи со случайной выборкой в упр. 3.4.2–5.) Полагая $\alpha = N/M$, точную формулу для C'_N можно заменить приближенной:

$$\frac{1}{1-\alpha} = 1+\alpha+\alpha^2+\alpha^3+\cdots. \tag{33}$$

Ее можно интерпретировать примерно так: с вероятностью α требуется более одной пробы, с вероятностью α^2 — более двух и т. д. Соответствующее среднее число проб при успешном поиске равно

$$C_N = \frac{1}{N}\sum_{k=0}^{N-1} C'_k = \frac{M+1}{N}\Big(\frac{1}{M+1}+\frac{1}{M}+\cdots+\frac{1}{M-N+2}\Big)$$
$$= \frac{M+1}{N}\big(H_{M+1}-H_{M-N+1}\big) \approx \frac{1}{\alpha}\ln\frac{1}{1-\alpha}. \tag{34}$$

Как отмечалось выше, многочисленные тесты показали, что алгоритм D с двумя независимыми хеш-функциями ведет себя, по сути, так же, как при равномерном исследовании для всех практических целей. В действительности двойное хеширование асимптотически эквивалентно равномерному исследованию в пределе при $M \to \infty$ (см. упр. 70).

Этим завершается анализ равномерного исследования. Для изучения линейного исследования и других типов разрешения коллизий следует строить более реалистичную теорию. Вероятностная модель, которая будет использоваться для этой цели, предполагает, что все M^N возможных "хеш-последовательностей"

$$a_1\, a_2 \ldots a_N, \qquad 0 \le a_j < M, \tag{35}$$

равновероятны. Здесь a_j обозначает начальный хеш-адрес j-го ключа, вставленного в таблицу. Среднее количество проб в случае успешного поиска при работе по данному алгоритму поиска, как и выше, обозначим через C_N; это значение представляет собой среднее количество проб, которые необходимы для поиска k-го ключа, усредненное по всем $1 \le k \le N$ при равновероятных ключах и по всем хеш-последовательностям (35) (все последовательности считаются равновероятными). Аналогично среднее количество проб, необходимых при вставке N-го ключа, считая все последовательности (35) равновероятными, обозначим как C'_{N-1}; эта величина представляет собой среднее количество проб в случае неудачного поиска, начинающегося при наличии в таблице $N-1$ элемента. При использовании открытой адресации

$$C_N = \frac{1}{N}\sum_{k=0}^{N-1} C'_k, \tag{36}$$

так что можно вывести одну величину из другой, как в (34).

Строго говоря, даже эта уточненная модель не лишена недостатков. Первый из них заключается в том, что не все хеш-последовательности равновероятны, поскольку сами ключи различны. Это делает вероятность того, что $a_1 = a_2$, немного меньше, чем $1/M$; однако отличие обычно незначительно, поскольку количество всех различных ключей, как правило, гораздо больше, чем M (см. упр. 24). Более того, хорошая хеш-функция способна использовать неслучайность типичных данных, дополнительно уменьшая вероятность того, что $a_1 = a_2$; таким образом, оценки числа проб будут пессимистическими. Другая неточность модели указана на рис. 43: ключи, встретившиеся ранее (за небольшими исключениями), ищутся чаще других. Таким образом, оценки C_N окажутся дважды пессимистичными и на практике алгоритмы будут иметь бóльшую производительность, чем предсказано в результате нашего анализа.

С учетом этих предостережений можно приступать к "точному" анализу линейного исследования*. Пусть $f(M,N)$ — число хеш-последовательностей (35), таких, что позиция 0 таблицы будет пуста после вставки ключей с помощью алгоритма L. Из циклической симметрии линейного исследования следует, что позиция 0 пуста так же часто, как и любая другая позиция, поэтому она пуста с вероятностью $1 - N/M$; другими словами

$$f(M,N) = \Big(1 - \frac{N}{M}\Big)M^N. \tag{37}$$

По определению также полагаем, что $f(0,0) = 1$. Пусть $g(M,N,k)$ — число хеш-последовательностей (35), таких, что алгоритм оставляет позицию 0 пустой, позиции от 1 до k занятыми и позицию $k+1$ — пустой. Имеем

$$g(M,N,k) = \binom{N}{k} f(k+1,k) f(M-k-1,N-k), \tag{38}$$

поскольку все такие хеш-последовательности состоят из двух подпоследовательностей. Первая (в ней содержится k элементов $a_i \le k$) оставляет позицию 0 пустой, а позиции от 1 до k занятыми; вторая (в ней содержится $N-k$ элементов $a_j \ge k+1$) оставляет позицию $k+1$ пустой. Существует $f(k+1,k)$ последовательностей первого типа, $f(M-k-1,N-k)$ последовательностей второго типа и $\binom{N}{k}$ способов их смешения. И, наконец, положим P_k равным вероятности того, что при вставке $(N+1)$-го ключа требуется в точности $k+1$ проб. Отсюда следует (см. упр. 25), что

$$P_k = M^{-N}\big(g(M,N,k) + g(M,N,k+1) + \cdots + g(M,N,N)\big). \tag{39}$$

Теперь $C'_N = \sum_{k=0}^{N}(k+1)P_k$; с учетом формул (36)–(39) и после определенных упрощений получим следующий результат.

---

* Автор не может не сделать вставку автобиографического характера: я впервые сформулировал предлагаемый вывод в 1962 году, вскоре после начала работы над "*Искусством программирования*". Поскольку это был первый успешно проанализированный мною нетривиальный алгоритм, он оказал сильное влияние на структуру этих книг. Даже сейчас анализ алгоритмов остается одним из главных занятий в моей жизни.

Теорема K. *Среднее количество проб, необходимых при использовании алгоритма L в предположении равновероятности всех M^N хеш-последовательностей* (35), *составляет*

$$C_N = \tfrac{1}{2}\big(1 + Q_0(M, N-1)\big) \quad \text{(успешный поиск)}, \tag{40}$$

$$C'_N = \tfrac{1}{2}\big(1 + Q_1(M, N)\big) \quad \text{(неудачный поиск)}, \tag{41}$$

где

$$\begin{aligned} Q_r(M,N) &= \binom{r}{0} + \binom{r+1}{1}\frac{N}{M} + \binom{r+2}{2}\frac{N(N-1)}{M^2} + \cdots \\ &= \sum_{k\ge 0}\binom{r+k}{k}\frac{N}{M}\frac{N-1}{M}\cdots\frac{N-k+1}{M}. \end{aligned} \tag{42}$$

Доказательство. Подробные вычисления проведены в упр. 27 (информацию об отклонении можно найти в упр. 28, 67 и 68). ▌

Странная функция $Q_r(M,N)$, которая появилась в теореме, только выглядит страшно; с ней вполне можно работать. Имеем

$$N^k - \binom{k}{2}N^{k-1} \le N(N-1)\ldots(N-k+1) \le N^k;$$

следовательно, если $N/M = \alpha$, то

$$\sum_{k\ge 0}\binom{r+k}{k}\left(N^k - \binom{k}{2}N^{k-1}\right)\Big/M^k \le Q_r(M,N) \le \sum_{k\ge 0}\binom{r+k}{k}N^k/M^k,$$

$$\sum_{k\ge 0}\binom{r+k}{k}\alpha^k - \frac{\alpha}{M}\sum_{k\ge 0}\binom{r+k}{k}\binom{k}{2}\alpha^{k-2} \le Q_r(M,\alpha M) \le \sum_{k\ge 0}\binom{r+k}{k}\alpha^k;$$

т. е.

$$\frac{1}{(1-\alpha)^{r+1}} - \frac{1}{M}\binom{r+2}{2}\frac{\alpha}{(1-\alpha)^{r+3}} \le Q_r(M,\alpha M) \le \frac{1}{(1-\alpha)^{r+1}}. \tag{43}$$

Это соотношение дает хорошую оценку $Q_r(M,N)$ при больших M и α, не слишком близком к 1 (нижняя оценка более точная, чем верхняя). При $\alpha \to 1$ эти формулы становятся непригодными, но, к счастью, $Q_0(M, M-1)$ представляет собой функцию $Q(M)$, асимптотическое поведение которой детально рассматривалось в разделе 1.2.11.3; $Q_1(M, M-1)$ просто равно M (см. упр. 50). В стандартных обозначениях гипергеометрических функций 1.2.6–39 имеем $Q_r(M,N) = F(r+1, -N, 1, -1/M) = F\left(\begin{smallmatrix} r+1, -N, 1 \\ 1 \end{smallmatrix} \,\middle|\, -\frac{1}{M}\right)$*.

Другой подход к анализу линейного исследования был предложен Г. Шеем (мл.) (G. Schay, Jr.) [*CACM* **5** и В. Г. Спрутом (W. G. Spruth) (1962), 459–462]. Хотя их метод дает только приближение к точным формулам теоремы K, он проливает дополнительный свет на алгоритм, а потому мы вкратце изложим его. Сперва рассмотрим удивительное свойство линейного исследования, впервые отмеченное В. В. Петерсоном (W. W. Peterson) в 1957 году.

---

* $F(x,y,a,b) = \sum_{k=0}^{\infty}\frac{x^{\bar{k}}\cdot y^{\bar{k}}\cdot b^k}{a^{\bar{k}}\cdot k!} = 1 + \frac{xy}{a}b + \frac{1}{2!}\frac{x(x+1)y(y+1)}{a(a+1)}b^2 + \ldots$ — стандартное обозначение для гипергеометрической функции. — *Прим. перев.*

Теорема Р. *Среднее количество проб при успешном поиске с помощью алгоритма L не зависит от порядка, в котором ключи были вставлены; он зависит только от числа ключей с тем или иным хеш-адресом.*

Другими словами, любое переупорядочение хеш-последовательности $a_1\, a_2 \ldots a_N$ дает хеш-последовательность с тем же средним смещением ключей от их хеш-адресов. (Предполагается, как упоминалось ранее, что все ключи в таблице одинаково важны. Если обращение к одним ключам происходит чаще, чем к другим, то, используя метод доказательства теоремы 6.1S, можно показать, что оптимальное расположение ключей достигается при их вставке в порядке уменьшения частот.)

Доказательство. Достаточно показать, что общее количество проб, необходимых для вставки ключей для хеш-последовательности $a_1\, a_2 \ldots a_N$, такое же, как и для последовательности $a_1 \ldots a_{i-1}\, a_{i+1}\, a_i\, a_{i+2} \ldots a_N$, $1 \le i < N$. Очевидно, что нет никаких отличий между этими двумя случаями, пока $(i+1)$-й ключ второй последовательности не попадает в позицию, занимаемую i-м ключом в первой последовательности. Однако тогда i- и $(i+1)$-й элементы просто меняются местами, так что количество проб для $(i+1)$-го ключа уменьшается на столько же, на сколько увеличивается число для i-го ключа. ▌

Согласно теореме Р средняя длина поиска для хеш-последовательности $a_1\, a_2 \ldots a_N$ может быть определена из чисел $b_0\, b_1 \ldots b_{M-1}$, где b_j представляет собой количество элементов a_k, равных j. По этой последовательности можно определить "последовательность переносов" $c_0\, c_1 \ldots c_{M-1}$, где c_j равно числу ключей, для которых при вставке ключа проверяются обе позиции (j и $j-1$). Эта последовательность определяется правилом

$$c_j = \begin{cases} 0, & \text{если } b_j = c_{(j+1) \bmod M} = 0; \\ b_j + c_{(j+1) \bmod M} - 1 & \text{в противном случае.} \end{cases} \tag{44}$$

Например, пусть $M = 10$, $N = 8$ и $b_0 \ldots b_9 = 0\ 3\ 2\ 0\ 1\ 0\ 0\ 0\ 0\ 2$; тогда $c_0 \ldots c_9 = 2\ 3\ 1\ 0\ 0\ 0\ 0\ 1\ 2\ 3$, поскольку один ключ должен быть перенесен из позиции 2 в позицию 1, три — из позиции 1 в позицию 0, два — из позиции 0 в позицию 9 и т. д. Имеем $b_0 + b_1 + \cdots + b_{M-1} = N$, и среднее количество проб, необходимых для выборки N ключей, составляет

$$1 + (c_0 + c_1 + \cdots + c_{M-1})/N. \tag{45}$$

Кажется, что правило (44) представляет собой циклическое определение чисел c через самих себя, но в действительности при любом $N < M$ система имеет единственное решение (см. упр. 32).

Шей и Спрут использовали эту идею для определения вероятности q_k того, что $c_j = k$ через вероятности p_k того, что $b_j = k$. (Эти вероятности не зависят от j.) Таким образом,

$$\begin{aligned} q_0 &= p_0 q_0 + p_1 q_0 + p_0 q_1, \\ q_1 &= p_2 q_0 + p_1 q_1 + p_0 q_2, \\ q_2 &= p_3 q_0 + p_2 q_1 + p_1 q_2 + p_0 q_3 \end{aligned} \tag{46}$$

и т. д., поскольку, например, вероятность того, что $c_j = 2$, представляет собой вероятность того, что $b_j + c_{(j+1) \bmod M} = 3$. Пусть $B(z) = \sum p_k z^k$ и $C(z) = \sum q_k z^k$ —

производящие функции для этих распределений вероятностей; уравнения (46) эквивалентны формуле

$$B(z)C(z) = p_0q_0 + (q_0 - p_0q_0)z + q_1z^2 + \cdots = p_0q_0(1-z) + zC(z).$$

Поскольку $B(1) = 1$, можно записать $B(z) = 1 + (z-1)D(z)$. Отсюда вытекает, что

$$C(z) = \frac{p_0q_0}{1-D(z)} = \frac{1-D(1)}{1-D(z)}, \tag{47}$$

так как $C(1) = 1$. Среднее количество проб, необходимых для выборки, в соответствии с (45) составляет

$$1 + \frac{M}{N}C'(1) = 1 + \frac{M}{N}\frac{D'(1)}{1-D(1)} = 1 + \frac{M}{2N}\frac{B''(1)}{1-B'(1)}. \tag{48}$$

В связи с тем, что предполагается равновероятность всех хеш-последовательностей $a_1 \ldots a_N$, имеем

$$\begin{aligned} p_k &= \Pr(\text{для фиксированного } j \text{ в точности } k \text{ из } a_i \text{ равны } j) \\ &= \binom{N}{k}\left(\frac{1}{M}\right)^k\left(1-\frac{1}{M}\right)^{N-k}; \end{aligned} \tag{49}$$

следовательно,

$$B(z) = \left(1 + \frac{z-1}{M}\right)^N, \qquad B'(1) = \frac{N}{M}, \qquad B''(1) = \frac{N(N-1)}{M^2} \tag{50}$$

и среднее количество проб согласно (48) составит

$$C_N = \frac{1}{2}\left(1 + \frac{M-1}{M-N}\right). \tag{51}$$

В состоянии ли читатель указать неверные рассуждения, вызвавшие отличие между полученным ответом и точным результатом в теореме K? (См. упр. 33.)

***Анализ оптимальности.** Мы рассмотрели несколько примеров последовательностей проб для открытой адресации, в результате чего, естественно, возникает вопрос, какая из них *наилучшая из возможных* в некотором разумном смысле. Эта задача имеет интересную постановку, предложенную Д. Д. Ульманом (J. D. Ullman) [*JACM* **19** (1972), 569–575]: вместо того чтобы вычислять хеш-адрес $h(K)$, мы отображаем каждый ключ K на перестановку множества $\{0, 1, \ldots, M-1\}$, которая представляет последовательность проб при использовании K. Каждой из $M!$ перестановок назначена вероятность, и предполагается, что обобщенная хеш-функция выбирает каждую перестановку с этой вероятностью. Вопрос формулируется следующим образом: "Какое назначение вероятностей данным перестановкам даст наивысшую производительность, т. е. минимизирует соответствующие средние числа проб C_N и C'_N?"

Например, если назначить каждой перестановке вероятность $1/M!$, то получится *равномерное исследование*, проанализированное в (32) и (34). Однако Ульманом был найден пример с $M = 4$ и $N = 2$, для которого C'_N меньше, чем значение

$\frac{5}{3}$, полученное для равномерного исследования. Его построение назначает нулевые значения всем перестановкам, кроме следующих шести.

Перестановка	Вероятность	Перестановка	Вероятность	
0 1 2 3	$(1+2\epsilon)/6$	1 0 3 2	$(1+2\epsilon)/6$	(52)
2 0 1 3	$(1-\epsilon)/6$	2 1 0 3	$(1-\epsilon)/6$	
3 0 1 2	$(1-\epsilon)/6$	3 1 0 2	$(1-\epsilon)/6$	

Грубо говоря, на первом шаге предпочтение отдается 2 или 3, а вторая проба всегда проверяет 0 или 1. Среднее количество проб, необходимых для вставки третьего элемента, C_3', равно $\frac{5}{3}-\frac{1}{9}\epsilon+O(\epsilon^2)$, так что можно улучшить равномерное исследование, присвоив ϵ малое положительное значение.

Однако соответствующее значение C_1' для этих вероятностей составляет $\frac{23}{18}+O(\epsilon)$, что больше $\frac{5}{4}$ (значение для равномерного исследования). Ульман доказал, что любое назначение вероятностей, такое, что $C_N' < (M+1)/(M+1-N)$, для некоторого N всегда влечет справедливость $C_n' > (M+1)/(M+1-n)$ для некоторого $n < N$; нельзя все время побеждать равномерное хеширование...

В действительности количество проб при *успешном* поиске является лучшим критерием, чем C_N'. Перестановки в (52) не приводят к улучшению значения C_N для любых N, и Ульман предположил, что никакие назначения вероятностей не сделают C_N меньше, чем для равномерного исследования: $\big((M+1)/N\big)(H_{M+1}-H_{M+1-N})$. Эндрю Яо (Andrew Yao) доказал асимптотическую форму этого утверждения, показав, что предельная цена при $N=\alpha M$ и $M\to\infty$ всегда $\geq \frac{1}{\alpha}\ln\frac{1}{1-\alpha}$ [*JACM* **32** (1985), 687–693].

Строгую форму предположения Ульмана очень сложно доказать, в особенности потому, что существует много вариантов назначения вероятностей для получения эффекта равномерного исследования; нет необходимости назначать вероятности $1/M!$ каждой перестановке. Например, следующие назначения также дают эквивалент равномерного исследования.

Перестановка	Вероятность	Перестановка	Вероятность	
0 1 2 3	1/6	0 2 1 3	1/12	
1 2 3 0	1/6	1 3 2 0	1/12	(53)
2 3 0 1	1/6	2 0 3 1	1/12	
3 0 1 2	1/6	3 1 0 2	1/12	

(Остальным шестнадцати перестановкам назначена нулевая вероятность.)

Следующая теорема характеризует *все* присвоения, дающие поведение равномерного исследования.

Теорема U. *Назначив вероятности перестановкам, все* $\binom{M}{N}$ *конфигурации пустых и занятых ячеек можно сделать равновероятными после* N *вставок для* $0<N<M$ *тогда и только тогда, когда сумма вероятностей, назначенная всем перестановкам, первые* N *элементов которых являются членами данного* N*-элементного множества, равна* $1/\binom{M}{N}$ *для всех* N *и для всех* N*-элементных множеств.*

Например, сумма вероятностей, назначенных каждой из $3!(M-3)!$ перестановок, которые начинаются числами $\{0,1,2\}$ в заданном порядке, должна составлять $1/\binom{M}{3}=3!(M-3)!/M!$. Заметьте, что (53) удовлетворяет условию теоремы, поскольку $1/6+1/12=1/4$.

Доказательство. Пусть $A \subseteq \{0, 1, \ldots, M-1\}$ и пусть $\Pi(A)$ — множество всех перестановок, первые $|A|$ элементов которых являются членами A. Обозначим также через $S(A)$ сумму вероятностей, назначенных этим перестановкам. Пусть $P_k(A)$ — вероятность того, что первые $|A|$ вставок при помощи процедуры открытой адресации займут места, определенные множеством A, и последняя вставка при этом потребует ровно k проб. И, наконец, пусть $P(A) = P_1(A) + P_2(A) + \cdots$. Доказательство проводится по индукции по $N \geq 1$. Предположим, что

$$P(A) = S(A) = 1\Big/\binom{M}{N}$$

для всех множеств A с $|A| = n < N$. Пусть B — некоторое N-элементное множество. Тогда

$$P_k(B) = \sum_{\substack{A \subseteq B \\ |A|=k}} \sum_{\pi \in \Pi(A)} \Pr(\pi) P\big(B \setminus \{\pi_k\}\big),$$

где $\Pr(\pi)$ представляет собой вероятность, назначенную перестановке π, и π_k — ее k-й элемент. По индукции

$$P_k(B) = \sum_{\substack{A \subseteq B \\ |A|=k}} \frac{1}{\binom{M}{N-1}} \sum_{\pi \in \Pi(A)} \Pr(\pi),$$

что равняется

$$\binom{N}{k}\Big/\binom{M}{N-1}\binom{M}{k} \qquad \text{при } k < N.$$

Следовательно,

$$P(B) = \frac{1}{\binom{M}{N-1}}\left(S(B) + \sum_{k=1}^{N-1} \frac{\binom{N}{k}}{\binom{M}{k}}\right),$$

а это может быть равно $1/\binom{M}{N}$ тогда и только тогда, когда $S(B)$ имеет корректное значение. ∎

Внешний поиск. Технология хеширования вполне подходит для внешнего поиска на устройствах хранения с прямым доступом наподобие дисков или барабанов. Для таких приложений, как и в разделе 6.2.4, желательно минимизировать количество обращений к файлу. Это условие двояко влияет на выбор алгоритмов.

1) Разумно потратить больше времени на вычисление хеш-функции, поскольку потери из-за плохого хеширования гораздо больше цены дополнительного времени, требуемого для аккуратного выполнения вычислений.

2) Записи обычно группируются в *страницы* или *блоки* с тем, чтобы за один раз извлекать из внешней памяти несколько записей.

Файл делится на M блоков по b записей. Коллизии при этом не вызывают никаких проблем до тех пор, пока одинаковые хеш-адреса имеют не более b ключей. Похоже, что наилучшими методами разрешения коллизий являются следующие.

A) *Использование цепочек с раздельными списками.* Если в один и тот же блок попадает более b записей, в конце первого блока можно вставить ссылку на переполняющую запись. Такие переполняющие записи содержатся в специальной

области переполнения. Обычно не имеет смысла разбивать область переполнения на блоки, поскольку, как правило, возникает сравнительно немного переполнений; таким образом, "лишние" записи связываются одна с другой, поэтому $(b+k)$-я запись списка требует $1+k$ обращений. Обычно стóит оставлять место для переполнений в каждом цилиндре дискового файла, чтобы как можно больше обращений оставалось в пределах одного и того же цилиндра.

Хотя этот метод обработки переполнений кажется неэффективным, статистически число переполнений достаточно мало для того, чтобы среднее время поиска оставалось очень хорошим. В табл. 2 и 3 показаны средние количества требуемых обращений как функций коэффициента заполненности

$$\alpha = N/Mb \tag{54}$$

для фиксированного α и $M,N \to \infty$. Любопытно, что при $\alpha = 1$ асимптотическое количество обращений для неудачного поиска увеличивается с ростом b.

Таблица 2

СРЕДНЕЕ КОЛИЧЕСТВО ОБРАЩЕНИЙ В СЛУЧАЕ НЕУДАЧНОГО ПОИСКА ПО РАЗДЕЛЬНЫМ ЦЕПОЧКАМ

Размер	Коэффициент заполнения, α									
блока, b	10%	20%	30%	40%	50%	60%	70%	80%	90%	95%
1	1.0048	1.0187	1.0408	1.0703	1.1065	1.1488	1.197	1.249	1.307	1.34
2	1.0012	1.0088	1.0269	1.0581	1.1036	1.1638	1.238	1.327	1.428	1.48
3	1.0003	1.0038	1.0162	1.0433	1.0898	1.1588	1.252	1.369	1.509	1.59
4	1.0001	1.0016	1.0095	1.0314	1.0751	1.1476	1.253	1.394	1.571	1.67
5	1.0000	1.0007	1.0056	1.0225	1.0619	1.1346	1.249	1.410	1.620	1.74
10	1.0000	1.0000	1.0004	1.0041	1.0222	1.0773	1.201	1.426	1.773	2.00
20	1.0000	1.0000	1.0000	1.0001	1.0028	1.0234	1.113	1.367	1.898	2.29
50	1.0000	1.0000	1.0000	1.0000	1.0000	1.0007	1.018	1.182	1.920	2.70

Таблица 3

СРЕДНЕЕ КОЛИЧЕСТВО ОБРАЩЕНИЙ В СЛУЧАЕ УСПЕШНОГО ПОИСКА ПО РАЗДЕЛЬНЫМ ЦЕПОЧКАМ

Размер	Коэффициент заполнения, α									
блока, b	10%	20%	30%	40%	50%	60%	70%	80%	90%	95%
1	1.0500	1.1000	1.1500	1.2000	1.2500	1.3000	1.350	1.400	1.450	1.48
2	1.0063	1.0242	1.0520	1.0883	1.1321	1.1823	1.238	1.299	1.364	1.40
3	1.0010	1.0071	1.0215	1.0458	1.0806	1.1259	1.181	1.246	1.319	1.36
4	1.0002	1.0023	1.0097	1.0257	1.0527	1.0922	1.145	1.211	1.290	1.33
5	1.0000	1.0008	1.0046	1.0151	1.0358	1.0699	1.119	1.186	1.268	1.32
10	1.0000	1.0000	1.0002	1.0015	1.0070	1.0226	1.056	1.115	1.206	1.27
20	1.0000	1.0000	1.0000	1.0000	1.0005	1.0038	1.018	1.059	1.150	1.22
50	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.001	1.015	1.083	1.16

B) *Использование срастающихся цепочек.* Можно не выделять специальную область переполнения, а адаптировать алгоритм C для работы с внешними файлами. Для каждого цилиндра может поддерживаться двусвязный список свободного пространства, связывающий совместно все не полностью заполненные блоки. При исполь-

зовании этой схемы в каждом блоке содержится счетчик, указывающий количество пустых позиций записей. Такой блок удаляется из двусвязного списка только по достижении счетчиком нулевого значения. Для распределения переполнений может применяться "блуждающий указатель" (см. упр. 2.5–6) с тем, чтобы различные цепочки использовали различные блоки переполнения. Этот метод все еще не проанализирован, но может быть весьма полезен.

C) *Использование открытой адресации.* Можно обойтись и без ссылок и использовать "открытый" метод. При рассмотрении внешнего поиска линейное исследование, вероятно, лучше случайных проб, потому что величина c зачастую может быть выбрана таким образом, чтобы минимизировать скрытые задержки между последовательными доступами. Приближенная теоретическая модель линейного исследования, с которой мы работали ранее, может быть обобщена для учета влияния блоков. Она показывает, что линейное исследование в самом деле вполне удовлетворительно, пока таблица не слишком заполнена. Например, взгляните на табл. 4: при коэффициенте заполненности, равном 90%, и размере блока, равном 50, среднее количество обращений при успешном поиске равно всего лишь 1.04. Это даже *лучше*, чем требуемые при использовании метода цепочек (A) с тем же размером блока 1.08 обращения!

Таблица 4

СРЕДНЕЕ КОЛИЧЕСТВО ОБРАЩЕНИЙ В СЛУЧАЕ УСПЕШНОГО ПОИСКА ПРИ ЛИНЕЙНОМ ИССЛЕДОВАНИИ

Размер	Коэффициент заполнения, α									
блока, b	10%	20%	30%	40%	50%	60%	70%	80%	90%	95%
1	1.0556	1.1250	1.2143	1.3333	1.5000	1.7500	2.167	3.000	5.500	10.50
2	1.0062	1.0242	1.0553	1.1033	1.1767	1.2930	1.494	1.903	3.147	5.64
3	1.0009	1.0066	1.0201	1.0450	1.0872	1.1584	1.286	1.554	2.378	4.04
4	1.0001	1.0021	1.0085	1.0227	1.0497	1.0984	1.190	1.386	2.000	3.24
5	1.0000	1.0007	1.0039	1.0124	1.0307	1.0661	1.136	1.289	1.777	2.77
10	1.0000	1.0000	1.0001	1.0011	1.0047	1.0154	1.042	1.110	1.345	1.84
20	1.0000	1.0000	1.0000	1.0000	1.0003	1.0020	1.010	1.036	1.144	1.39
50	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.001	1.005	1.040	1.13

Анализ методов A и C влечет очень интересные с математической точки зрения выводы; здесь мы приведем лишь некоторые результаты, поскольку подробности приводятся в упр. 49 и 55. Формулы содержат две функции, тесно связанные с Q-функциями из теоремы K, а именно:

$$R(\alpha,n) = \frac{n}{n+1} + \frac{n^2\alpha}{(n+1)(n+2)} + \frac{n^3\alpha^2}{(n+1)(n+2)(n+3)} + \cdots \tag{55}$$

и

$$\begin{aligned} t_n(\alpha) &= e^{-n\alpha}\left(\frac{(\alpha n)^n}{(n+1)!} + 2\,\frac{(\alpha n)^{n+1}}{(n+2)!} + 3\,\frac{(\alpha n)^{n+2}}{(n+3)!} + \cdots\right) \\ &= \frac{e^{-n\alpha}n^n\alpha^n}{n!}\big(1-(1-\alpha)R(\alpha,n)\big). \end{aligned} \tag{56}$$

С помощью этих функций среднее число обращений, выполняемых при неудачном поиске по методу A, выражается как

$$C'_N = 1 + \alpha b t_b(\alpha) + O\Big(\frac{1}{M}\Big) \tag{57}$$

при $M,N \to \infty$, а соответствующее число при успешном поиске может быть записано как

$$C_N = 1 + \frac{e^{-b\alpha} b^b \alpha^b}{2b!}\big(2 + (\alpha - 1)b + \big(\alpha^2 + (\alpha-1)^2(b-1)\big)R(\alpha, b)\big) + O\Big(\frac{1}{M}\Big). \tag{58}$$

Предельные значения соответствуют величинам, приведенным в табл. 2 и 3.

Поскольку метод цепочек (A) требует отдельной области переполнения, необходимо оценить количество возможных переполнений. Среднее число переполнений составляет $M(C'_N - 1) = N t_b(\alpha)$, так как $C'_N - 1$ — среднее число переполнений в любом данном списке. Таким образом, табл. 2 может использоваться для нахождения требуемого размера области переполнения. Для фиксированного α стандартное отклонение общего количества переполнений будет примерно пропорционально $\sqrt{M}$ при $M \to \infty$.

Асимптотические значения C'_N и C_N приведены в упр. 53, однако эти приближения не очень хороши при малых b или больших α. К счастью, ряд для $R(\alpha, n)$ сходится очень быстро даже при больших α, так что формулы могут быть вычислены с любой желательной точностью без больших трудностей. Максимальное значение достигается при $\alpha = 1$, когда при $b \to \infty$

$$\max C'_N = 1 + \frac{e^{-b} b^{b+1}}{b!} = \sqrt{\frac{b}{2\pi}} + 1 + O(b^{-1/2}), \tag{59}$$

$$\max C_N = 1 + \frac{e^{-b} b^b}{2b!}\big(R(b) + 1\big) = \frac{5}{4} + \sqrt{\frac{2}{9\pi b}} + O(b^{-1}) \tag{60}$$

согласно приближению Стирлинга и анализу функции $R(n) = R(1, n) - 1$ (см. раздел 1.2.11.3).

Среднее количество обращений при успешном внешнем поиске с помощью *линейного* исследования имеет необыкновенно простой вид:

$$C_N \approx 1 + t_b(\alpha) + t_{2b}(\alpha) + t_{3b}(\alpha) + \cdots. \tag{61}$$

Эту формулу можно пояснить следующим образом: общее среднее количество обращений при поиске всех N ключей равно NC_N, что представляет собой $N + T_1 + T_2 + \cdots$, где T_k — среднее количество ключей, требующих более k обращений. Теорема P гласит, что ключи можно вставлять в любом порядке без влияния на C_N; отсюда следует, что T_k равно среднему числу переполняющих записей, которые имелись бы при использовании метода цепочек с M/k блоками размером kb, т. е. $N t_{kb}(\alpha)$, как и говорилось выше. Подробный анализ формулы (61) приводится в упр. 55.

Обсуждение практических вопросов построения внешних хеш-таблиц дано Чарльзом А. Ольсоном (Charles A. Olson) [*Proc. ACM Nat. Conf.* **24** (1969), 539–549]. В его работу включено несколько практических примеров; в ней указано, что количество переполняющих записей значительно увеличивается, если файл служит объектом частых вставок/удалений без перемещения записей. В работе также представлен анализ этой ситуации, выполненный совместно с Дж. А. де Пейстером (J. A. de Peyster).

Сравнение методов. Мы изучили много различных методов поиска, однако какой метод лучше выбрать для конкретного приложения? В нескольких словах невозможно изложить все, что хотелось бы учесть при выборе метода поиска; однако следующие соображения представляются наиболее важными в отношении скорости работы алгоритма и объема памяти, требуемой этими методами.

На рис. 44 представлены результаты анализов алгоритмов, проведенных в данном разделе, и показано, что различные методы разрешения коллизий приводят к различному числу проб. Однако количество проб — не единственный критерий, так как при использовании разных методов на проведение пробы требуется разное время, что существенно влияет на время работы в целом (см. рис. 42). При линейном исследовании приходится значительно чаще по сравнению с другими методами, показанными на рис. 44, обращаться к таблице, зато этот метод чрезвычайно прост. Кроме того, линейное исследование — не такой уж плохой метод: при заполненности таблицы на 90% алгоритм L требует в среднем меньше 5.5 проб для поиска в таблице случайного элемента (впрочем, при 90% заполненности таблицы алгоритм L нуждается в примерно 50.5 пробах для вставки каждого *нового* элемента).

Как показано на рис. 44, метод цепочек очень экономичен с точки зрения числа проб и не экономичен с точки зрения требуемой дополнительной памяти для хранения полей ссылок, что особенно неприятно при наличии малых записей. При этом более выгодным по сравнению с методом цепочек становится метод открытой адресации. Так, при выборе между таблицей с цепочками объемом 500 элементов и таблицей с открытой адресацией объемом 1 000 элементов последняя явно предпочтительнее, поскольку обеспечивает эффективный поиск среди 500 записей и вмещает в два раза больше данных. С другой стороны, иногда в силу размера записей или специфики используемого формата можно получить место для полей ссылок фактически бесплатно (см. упр. 65).

Как ведут себя методы хеширования по сравнению с другими методами поиска, изученными в этой главе? С точки зрения скорости работы они лучше других методов при большом количестве записей, поскольку среднее время поиска при хеш-методе остается ограниченным, когда $N \to \infty$, если таблица не становится слишком заполненной. Например, программа L требует всего лишь 55 единиц времени для успешного поиска при заполненной на 90% таблице, что быстрее самой быстрой MIX-программы (см. упр. 6.2.1–24) при $N > 600$, и цена этой скорости составляет 11% пространства. Более того, бинарный поиск годится только для фиксированных таблиц, в то время как хеш-таблицы позволяют делать эффективные вставки.

Можно также сравнить программу L с методами поиска, ориентированными на работу с деревьями, которые позволяют выполнять динамические вставки. Программа L при таблице, заполненной на 90%, работает быстрее, чем программа 6.2.2T при $N > 90$, и быстрее, чем программа 6.3D (упр. 6.3–9) при $N > 75$.

Только один метод из рассмотренных в данной главе эффективен в случае удачного поиска и не перерасходует память — это модификация Брента алгоритма D. Метод Брента позволяет поместить N записей в таблицу размером $M = N + 1$ и найти любую запись в среднем за 2.5 пробы. При этом не требуется дополнительное пространство для полей ссылок или битов дескрипторов; однако неудачный поиск может оказаться весьма медленным (для него может потребоваться около $N/2$ проб).

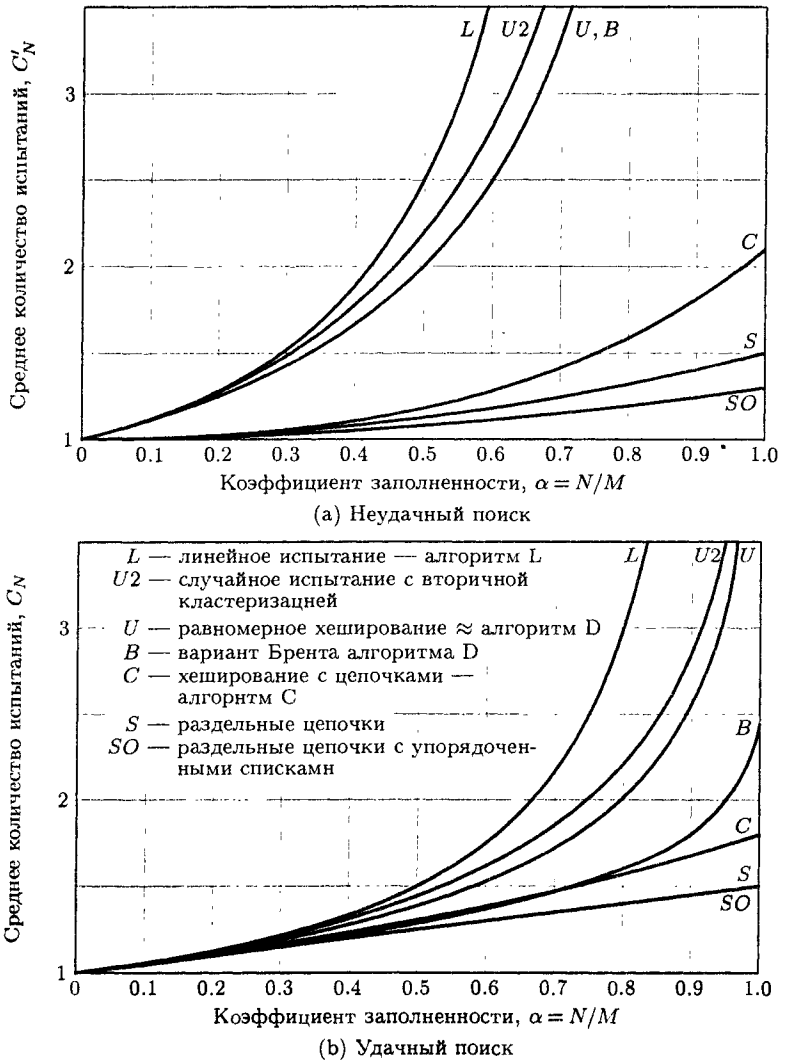


Рис. 44. Сравнение методов разрешения коллизий: предельные значения среднего количества проб при $M \to \infty$.

Таким образом, хеширование имеет ряд преимуществ. С другой стороны, есть три важных аспекта, по которым хеширование уступает рассмотренным ранее методам.

А) После неудачного поиска в хеш-таблице известно только то, что искомый ключ отсутствует в таблице, а методы, основанные на сравнении ключей, всегда дают дополнительную информацию, например позволяют найти наибольший ключ $\leq K$ и/или наименьший ключ $\geq K$*. Во многих приложениях получить такую информацию крайне важно; например, она позволяет интерполировать функцию

* Впрочем, поскольку мы говорим о неудачном поиске, приведенные автором неравенства становятся строгими. — *Прим. перев.*

по ее табличным значениям или использовать алгоритм, основанный на сравнении ключей для поиска всех ключей, лежащих *между* двумя данными значениями K и K'. Кроме того, алгоритмы поиска по дереву из раздела 6.2 позволяют легко представить содержимое таблицы в порядке возрастания без отдельной сортировки.

B) Часто очень сложно распределить память для хеш-таблицы — для нее следует заблаговременно выделить область памяти, и не всегда удается заранее сказать, какой размер необходим. Выделяя слишком много памяти, можно тем самым ограничить объем памяти для других списков или других пользователей; при выделении недостаточного количества памяти произойдет переполнение таблицы. В противоположность этому методу алгоритмы поиска и вставки по дереву никогда не увеличиваются сверх необходимого размера. В среде с виртуальной памятью можно локализовать обращения к памяти при выполнении поиска по дереву (или цифрового поиска по дереву), в то время как при создании большой хеш-таблицы необходимо обращаться к операционной системе, чтобы получить доступ к новой странице памяти почти всякий раз при хешировании очередного ключа.

C) Наконец, нужно свято верить в теорию вероятностей, применяя методы хеширования, поскольку они эффективны только в среднем, в то время как наихудшие случаи просто ужасны. Как и при использовании датчиков случайных чисел, никогда нельзя быть абсолютно уверенным в том, что хеш-функция будет удовлетворительно работать с новым множеством данных. Таким образом, хеширование — не самый подходящий метод для некоторых приложений реального времени (наподобие контроля за дорожным движением), когда на карту поставлены человеческие жизни. Алгоритмы сбалансированных деревьев из разделов 6.2.3 и 6.2.4 более безопасны, поскольку обеспечивают гарантированную верхнюю границу времени поиска.

История. Идея хеширования впервые была высказана Г. П. Ланом (H. P. Luhn) при написании внутреннего меморандума IBM в январе 1953 года с предложением использовать метод цепочек. В этом меморандуме фактически впервые предлагалось использовать связанные линейные списки при решении практических задач. Он указал, что для внешнего поиска желательно использовать блоки, содержащие более одного элемента. Вскоре после этого Э. Д. Лин (A. D. Lin) развил теорию Лана и предложил технологию для обработки переполнений с помощью "вырожденных адресов"; например, при переполнении первичного блока 2 748 переполняющих записей попадают во вторичный блок 274; переполнения из этого блока попадают в третичный блок 27 и т. д. (в предположении, что имеется 10 000 первичных блоков, 1 000 вторичных, 100 третичных и т. д.). Хеш-функция, предложенная Ланом, по своей природе была буквенно-цифровой, например комбинировались соседние пары цифр ключа при помощи суммирования по модулю 10, так что 31 415 926 упаковывалось до 4 548.

Примерно в то же время идея хеширования независимо возникла и у другой группы сотрудников IBM: Жини М. Амдал (Gene M. Amdahl), Элейн М. Боэм (Elaine M. Boehme), Н. Рочестера (N. Rochester) и Артура Л. Сэмюэла (Arthur L. Samuel). Они создавали программу на ассемблере для IBM 701. Для разрешения проблемы, связанной с коллизиями, Жини Амдал предложила использовать открытую адресацию с линейным исследованием.

В открытой печати хеширование впервые было описано Арнольдом И. Думи (Arnold I. Dumey) [*Computers and Automation* **5**, 12 (December, 1956), 6–9]. Он был

первым, кто упомянул идею деления на простое число и использование остатка в качестве хеш-адреса. В интересной статье Думи упоминает цепочки, но открытой адресации в ней нет. Советский математик А. П. Ершов в 1957 году независимо разработал метод линейной открытой адресации [*ДАН СССР* **118** (1958), 427–430]; он опубликовал эмпирические результаты по количеству проб, справедливо предположив, что среднее количество проб для успешного поиска < 2 при $N/M < 2/3$.

Классическая статья W. W. Peterson, *IBM J. Research & Development* **1** (1957), 130–146, была первой важной работой, посвященной поиску в больших файлах. Петерсон определил открытую адресацию в общем случае, проанализировал производительность равномерного исследования и привел многочисленные эмпирические статистические данные о поведении линейной открытой адресации с блоками различных размеров, указав на ухудшение производительности при удалении элементов. Другой всесторонний обзор предмета был опубликован шестью годами позже Вернером Букхольцем (Werner Buchholz) [*IBM Systems J.* **2** (1963), 86–111]. В его работе особый упор сделан на исследовании хеш-функций. Корректный анализ алгоритма L был впервые опубликован А. Г. Конхеймом (A. G. Konheim) и Б. Вейссом (B. Weiss) [*SIAM J. Appl. Math.* **14** (1966), 1266–1274], а также В. Поддерюгиным [*Wissenschaftliche Zeitschrift der Technischen Universität Dresden* **17** (1968), 1087–1089].

К этому времени линейное исследование было единственным типом схемы открытой адресации, описанной в литературе, хотя несколькими исследователями независимо была разработана другая схема, основанная на неоднократном случайном применении независимых хеш-функций (см. упр. 48). В течение нескольких последующих лет хеширование стало широко использоваться, хотя не было опубликовано никаких новых работ. Затем Роберт Моррис (Robert Morris) написал обзор по хешированию [*CACM* **11** (1968), 38–44], оказавший впоследствии большое влияние на работы в этом направлении. В обзоре впервые появилась идея случайного исследования с вторичной кластеризацией. Статья Морриса оказалась наброском дальнейших исследований в этой области, которые завершились созданием алгоритма D и его усовершенствованных версий.

Интересно отметить, что слово "хеширование" (hashing), по-видимому, ни разу не появилось в печати (в его современном значении) до 60-х годов, хотя к тому времени оно прочно заняло место в компьютерном жаргоне во всем мире. Впервые это слово, вероятно, было использовано в книге H. Hellerman, *Digital Computer System Principles* (New York: McGraw-Hill, 1967), 152. При работе над этим разделом автор изучил около 60 относящихся к хешированию документов, однако более раннее упоминание данного термина встретилось лишь один раз в неопубликованном меморандуме В. В. Петерсона, датированном 1967 годом. Получилось так, что, хотя глагол "хешировать" (to hash) стал стандартным термином в средине 60-х годов, никто не решался использовать в печати столь недостойное слово до 1967 года!*

---

* Становление нового термина — процесс непростой, и, пожалуй, уместно вспомнить, что еще совсем недавно вместо слова "компьютер" в нашей литературе использовалось в лучшем случае ЭВМ, вместо "винчестер" — НЖМД (для тех, кто не помнит этот термин, напомним, что он означает "накопитель на жестком магнитном диске"), вместо "дискета" — ГМД (гибкий магнитный диск)... — *Прим. перев.*

Последние разработки. Со времени первой публикации этой главы в 1972 году в области теории и практики хеширования было сделано немало новых достижений, хотя основные идеи остались полезными для использования в обычных приложениях. Например, в книге J. S. Vitter and W.-C. Chen, *Design and Analysis of Coalesced Hashing* (New York: Oxford Univ. Press, 1987) рассмотрены и проанализированы различные поучительные варианты алгоритма C.

С практической точки зрения наиболее важным открытием в области технологии хеширования со времен 70-х годов, вероятно, является метод Витольда Литвина (Witold Litwin), называемый *линейным хешированием* [*Proc. 6th International Conf. on Very Large Databases* (1980), 212–223]. Линейное хеширование, которое не имеет ничего общего с классической технологией линейного исследования, позволяет многим хеш-адресам расти и/или выступать в роли вставляемых и удаляемых элементов. Превосходное обсуждение линейного хеширования, включая сравнение с другими методами, можно найти в работах Per-Åke Larson, *CACM* **31** (1988), 446–457, W. G. Griswold and G. M. Townsend, *Software Practice & Exp.* **23** (1993), 351–367 (в последней работе рассмотрены усовершенствования метода для одновременного использования множества больших и/или маленьких таблиц). Линейное хеширование может также использоваться для огромных баз данных, распределенных между многими узлами в сети [см. Litwin, Neimat, and Schneider, *ACM Trans. Database Syst.* **21** (1996), 480–525]. Альтернативная схема, называемая *расширяемым хешированием* (*extendible hashing*) и имеющая то свойство, что для выборки любой записи требуется не более двух ссылок на внешние страницы, была предложена в то же время в работе R. Fagin, J. Nievergelt, N. Pippenger, and H. R. Strong [*ACM Trans. Database Syst.* **4** (1979), 315–344]. Как линейное, так и расширяемое хеширования предпочтительнее использования B-деревьев (см. раздел 6.2.4) в том случае, когда порядок ключей не имеет значения.

В области теоретической были разработаны более сложные методы, которые могут гарантировать максимальное время доступа $O(1)$ со средним временем на вставку и удаление $O(1)$ независимо от ключа. Более того, общее количество пространства, требуемого для работы алгоритма, ограничено некоторой константой, умноженной на количество элементов, имеющихся в настоящий момент в наличии, плюс некоторая другая константа*. Этот результат, основанный на идеях Фредмана (Fredman), Комлёса (Komlós) и Семереди (Szemerédi) [*JACM* **31** (1984), 538–544], получен в работе Dietzfelbinger, Karlin, Mehlhorn, Meyer auf der Heide, Rohnert, and Tarjan [*SICOMP* **23** (1994), 738–761].

УПРАЖНЕНИЯ

1. [*20*] Насколько мало́ и насколько велико может быть содержимое rI1 по достижении команды 9H, приведенной в табл. 1, в предположении, что каждый из байтов 1, 2, 3 ключа K содержат символы с кодами, меньшими 30?

2. [*20*] Найдите довольно известное английское слово, которого нет в табл. 1 и которое может быть добавлено в нее без изменения программы.

---

* Иначе говоря, не мудрствуя, можно записать, что требуемая для хранения память составляет $C_1 \cdot N + C_2$. — *Прим. перев.*

3. [*23*] Объясните, почему программу, приведенную в табл. 1, нельзя заменить более простой программой, которая начинается с пяти следующих ниже команд, ни при какой константе a, с учетом того, что разные ключи не могут иметь одинаковые хеш-адреса?

```
LD1  K(1:1)  или  LD1N  K(1:1)
LD2  K(2:2)  или  LD2N  K(2:2)
INC1 a,2
LD2  K(3:3)
J2Z  9F
```

4. [*M30*] Сколько гостей следует пригласить на вечеринку, чтобы вероятность появления *трех* человек с одним и тем же днем рождения была не менее $\frac{1}{2}$?

5. [*15*] Мистер Ламер писал компилятор FORTRAN с использованием десятичного компьютера MIX, и ему понадобилась таблица символов для хранения имен переменных в программе во время компиляции. Эти имена имеют ограниченную длину — не более десяти символов. Ламер решил использовать хеш-таблицу с $M = 100$, а для скорости работы использовал хеш-функцию $h(K) =$ *крайний левый байт* K. Насколько хороша его идея?

6. [*15*] Насколько разумно заменить две первые команды из (3) командами LDA K; ENTX 0?

7. [*HM30*] (*Полиномиальное хеширование.*) Назначение этого упражнения — рассмотреть построение полиномов $P(x)$, как в (10), которые преобразуют n-битовые ключи в m-битовые адреса так, что различные ключи, отличающиеся не более чем t бит, будут иметь различные хеш-адреса. По заданным значениям n и $t \le n$, и по данному целому k, такому, что n делит $2^k - 1$, построим полином степени m, где m является функцией n, t и k. (Обычно при необходимости n увеличивается, так что k может быть выбрано достаточно малым.)

Пусть S — минимальное множество целых чисел, таких, что $\{1, 2, \ldots, t\} \subseteq S$ и $(2j) \bmod n \in S$ для всех $j \in S$. Например, при $n = 15$, $k = 4$ и $t = 6$ имеем $S = \{1, 2, 3, 4, 5, 6, 8, 10, 12, 9\}$. Теперь определим полином $P(x) = \prod_{j \in S}(x - \alpha^j)$, где α — элемент порядка n конечного поля $\mathrm{GF}(2^k)$, а коэффициенты $P(x)$ вычисляются в этом поле. Степень m полинома $P(x)$ равна числу элементов в S. Поскольку, если α^j — корень $P(x)$, то и α^{2j} — корень $P(x)$, отсюда следует, что коэффициенты p_j полинома $P(x)$ удовлетворяют условию $p_i^2 = p_i$, т. е. они представляют собой 0 или 1.

Докажите, что если $R(x) = r_{n-1}x^{n-1} + \cdots + r_1 x + r_0$ представляет собой некоторый ненулевой многочлен по модулю 2 с не более чем t ненулевыми коэффициентами, то $R(x)$ не кратен $P(x)$ по модулю 2. (Отсюда следует, что соответствующая хеш-функция ведет себя, как и было обещано*.)

8. [*M34*] (*Теорема о трех длинах.*) Пусть θ — иррациональное число между 0 и 1, представление которого в виде регулярной непрерывной дроби в обозначениях раздела 4.5.3 есть $\theta = //a_1, a_2, a_3, \ldots//$. Пусть $q_0 = 0$, $p_0 = 1$, $q_1 = 1$, $p_1 = 0$ и $q_{k+1} = a_k q_k + q_{k-1}$, $p_{k+1} = a_k p_k + p_{k-1}$ при $k \ge 1$. Пусть $\{x\}$ означает $x \bmod 1 = x - \lfloor x \rfloor$, а $\{x\}^+$ — $x - \lceil x \rceil + 1$. Перенумеруем отрезки, получающиеся в результате последовательной вставки точек $\{\theta\}, \{2\theta\}, \{3\theta\}, \ldots$ в интервал $[0\,..\,1]$, в порядке их появления, причем первому отрезку присваивается номер 0. Докажите справедливость следующих утверждений. Интервал номер s длиной $\{t\theta\}$, где $t = rq_k + q_{k-1}$, $0 \le r < a_k$, k четно и $0 \le s < q_k$, имеет левую границу $\{s\theta\}$ и правую границу $\{(s+t)\theta\}^+$. Интервал номер s длиной $1 - \{t\theta\}$, где $t =$

---

* Взгляните с этой точки зрения на полиномы $x^{16}+x^{12}+x^5+1$ и $x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}+x^8+x^7+x^5+x^4+x^2+x^1+1$, определяющие CCITT CRC-16 и CRC-32 соответственно, и рассмотрите возможность использования CRC в качестве хеш-функции. — *Прим. перев.*

$rq_k + q_{k-1}$, $0 \le r < a_k$, k нечетно и $0 \le s < q_k$, имеет левую границу $\{(s+t)\theta\}$ и правую границу $\{s\theta\}^+$. Любое положительное целое число n может быть единственным образом представлено в виде $n = rq_k + q_{k-1} + s$ при $k \ge 1$, $1 \le r \le a_k$ и $0 \le s < q_k$. В этих обозначениях перед вставкой точки $\{n\theta\}$ имеется n интервалов:

первые s интервалов (с номерами $0, \ldots, s-1$) длиной $\{(-1)^k(rq_k + q_{k-1})\theta\}$;
первые $n - q_k$ интервалов (с номерами $0, \ldots, n - q_k - 1$) длиной $\{(-1)^{k+1}q_k\theta\}$;
последние $q_k - s$ интервалов (с номерами $s, \ldots, q_k - 1$) длиной $\{(-1)^k((r-1)q_k + q_{k-1})\theta\}^+$.

Операция вставки $\{n\theta\}$ приводит к удалению интервала номер s третьего типа и его преобразованию в интервал номер s первого типа и номер $n - q_k$ второго типа.

9. [*M30*] При последовательной вставке точек $\{\theta\}$, $\{2\theta\}$, ... в отрезок $[0\,..\,1]$ теорема S утверждает, что каждая новая точка всегда разбивает один из наибольших имеющихся интервалов. Если интервал $[a\,..\,c]$ разбит таким образом на две части, $[a\,..\,b]$ и $[b\,..\,c]$, назовем разбиение *плохим*, если одна из частей более чем в два раза больше другой, т. е. если $b - a > 2(c - b)$ или $c - b > 2(b - a)$.

Докажите, что при некотором n точка $\{n\theta\}$ дает плохое разбиение, за исключением случаев $\theta \bmod 1 = \phi^{-1}$ и $\theta \bmod 1 = \phi^{-2}$, при которых *никогда* не бывает плохих разбиений.

10. [*M38*] (Р. Л. Грэхем (R. L. Graham).) Докажите, что если $\theta, \alpha_1, \ldots, \alpha_d$ — действительные числа, причем $\alpha_1 = 0$, если $n_1, \ldots, n_d$ — положительные целые числа и если точки $\{n\theta + \alpha_j\}$ вставлены в интервал $[0\,..\,1]$ при $0 \le n < n_j$ и $1 \le j \le d$, то $n_1 + \cdots + n_d$ получающихся интервалов (возможно, пустых) имеют не более $3d$ различных длин.

11. [*16*] Как правило, поиск чаще завершается успешно, чем неудачно. Исходя из этого, насколько разумно заменить строки 12–13 программы C строками 10–11?

▸ **12.** [*21*] Покажите, что программа C может быть переписана так, что во внутреннем цикле останется только один условный переход. Сравните время работы модифицированной и исходной программ.

▸ **13.** [*24*] (*Сокращенные ключи.*) Пусть $h(K)$ представляет собой хеш-функцию, а $q(K)$ — функцию от K, такую, что K можно определить по данным $h(K)$ и $q(K)$. Например, в случае хеширования путем деления можно положить $h(K) = K \bmod M$, а $q(K) = \lfloor K/M \rfloor$; при мультипликативном же хешировании в качестве $h(K)$ можно взять старшие биты $(AK/w) \bmod 1$, а в качестве $q(K)$ — остальные биты.

Покажите, что при использовании цепочек без списков перекрытий достаточно хранить значение $q(K)$ вместо K для каждой записи. (Это позволяет сэкономить пространство, необходимое для полей ссылок.) Измените алгоритм C, чтобы он позволял работать с такими сокращенными ключами и можно было избежать использования списков перекрытий и вспомогательной памяти для хранения переполняющих записей.

14. [*24*] (Э. В. Элькок (E. W. Elcock).) Покажите, что большая хеш-таблица может *разделять память* с другими связанными списками. Пусть каждое слово области списка имеет двухбитовое поле TAG и два ссылочных поля (LINK и AUX), имеющих следующий смысл.

TAG(P) = 0 указывает на слово в списке свободного пространства, LINK(P) указывает на следующий элемент в списке, а AUX(P) не используется.

TAG(P) = 1 указывает на используемое слово, где P не является хеш-адресом никакого ключа из хеш-таблицы; другие поля слова в позиции P могут иметь любой формат.

TAG(P) = 2 указывает, что P представляет собой хеш-адрес, по меньшей мере, одного ключа, AUX(P) указывает на связанный список, определяемый всеми такими ключами, а LINK(P) указывает на другое слово в памяти списка. При каждом обращении к слову с TAG(P) = 2 во время обработки любого списка

мы присваиваем P ← LINK(P) несколько раз до достижения слова с TAG(P) ≤ 1. (Для эффективности можно также изменить предшествующие ссылки так, что не нужно будет пропускать одни и те же элементы снова и снова.)

Определите подходящий алгоритм для вставки и получения ключей из такой хеш-таблицы.

15. [*16*] Почему в алгоритмах L и D стóит сообщать о переполнении при $N = M - 1$, а не при $N = M$?

16. [*10*] В программе L говорится, что K не должно быть нулем. А вдруг на самом деле она работает при $K = 0$?

17. [*15*] Почему бы просто не определить $h_2(K) = h_1(K)$ в (25) при $h_1(K) \neq 0$?

▶ **18.** [*21*] Что лучше использовать для замены строк 10–13 программы D — (31) или (30)? Дайте ответ на основании средних значений A, $S1$ и C.

19. [*40*] Проверьте эмпирически воздействие ограничения области значений $h_2(K)$ в алгоритме D, так что: (a) $1 \leq h_2(K) \leq r$ при $r = 1, 2, 3, \ldots, 10$; (b) $1 \leq h_2(K) \leq \rho M$ при $\rho = \frac{1}{10}, \frac{2}{10}, \ldots, \frac{9}{10}$.

20. [*M25*] (Р. Крутар (R. Krutar).) Измените алгоритм D следующим образом для удаления хеш-функции $h_2(K)$: на шаге D3 установите $c \leftarrow 0$, а в начале шага D4 установите $c \leftarrow c + 1$. Докажите, что, если $M = 2^m$, соответствующая последовательность проб $h_1(K)$, $(h_1(K) - 1) \bmod M, \ldots, \left(h_1(K) - \binom{M}{2}\right) \bmod M$ будет перестановкой $\{0, 1, \ldots, M-1\}$. Если такой метод "квадратичных проб" запрограммировать для компьютера MIX, как будет выглядеть полученная программа по сравнению с программами, показанными на рис. 42, в предположении, что алгоритм ведет себя, как при случайном опробовании с вторичной кластеризацией?

▶ **21.** [*20*] Предположим, что необходимо удалить запись из таблицы, построенной согласно алгоритму D, помечая ее как удаленную, как было предложено в тексте. Следует ли при этом уменьшать значение переменной N, используемой для управления алгоритмом D?

22. [*27*] Докажите, что алгоритм R оставляет таблицу в таком виде, как будто KEY[i] никогда не был вставлен.

▶ **23.** [*33*] Разработайте алгоритм, аналогичный алгоритму R, для удаления элементов из хеш-таблицы с цепочками, построенной по алгоритму C.

24. [*M20*] Предположим, что множество всех возможных ключей имеет MP элементов, из которых ровно P имеют некоторый данный хеш-адрес. (На практике P очень велико; например, если ключи представляют собой произвольные десятизначные числа и если $M = 10^3$, то $P = 10^7$.) Положим $M \geq 7$ и $N = 7$. Если из множества всех возможных ключей выбраны семь различных, то чему будет равна точная вероятность того, что будет получена хеш-последовательность 1 2 6 2 1 6 1 (т. е. $h(K_1) = 1$, $h(K_2) = 2, \ldots, h(K_7) = 1$)? Выразите ответ в виде функции от M и P.

25. [*M19*] Объясните, почему верна формула (39).

26. [*M20*] Чему равно количество хеш-последовательностей $a_1 a_2 \ldots a_9$, дающих при использовании линейного исследования картину занятых ячеек, приведенную в (21)?

27. [*M27*] Завершите доказательство теоремы K. (*Указание.* Пусть

$$s(n, x, y) = \sum_k \binom{n}{k} (x + k)^{k+1} (y - k)^{n-k-1} (y - n);$$

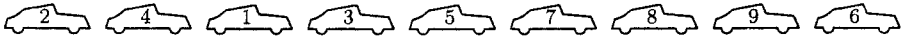
для доказательства того, что $s(n, x, y) = x(x + y)^n + ns(n-1, x+1, y-1)$, используйте биномиальную теорему Абеля (1.2.6–(16)).)

28. [*M30*] "В те времена укромные, теперь почти былинные*", когда компьютеры были *очень* медлительными, по скорости мигания лампочек на панели можно было увидеть, насколько быстро работает алгоритм L. Когда таблица заполнялась почти до конца, можно было заметить, что одни данные обрабатывались очень быстро, а другие — очень медленно.

Эти наблюдения наталкивают на мысль о большом значении стандартного отклонения количества проб в случае неудачного поиска при использовании линейного исследования. Найдите формулу, выражающую дисперсию через функции Q_r из теоремы K, и оцените ее значение при $N = \alpha M$ и $M \to \infty$.

29. [*M21*] (*Задача о парковке.*) Предположим, существует некоторая улица с односторонним движением, имеющая m мест для парковки, пронумерованных от 1 до m. В автомобиле рядом с водителем дремлет его жена, которая вдруг просыпается и требует немедленно остановиться. Муж послушно выполняет требование жены и останавливается на первом свободном месте для парковки. Однако, если "нет нигде стоянки, Брайтон весь забит", т. е. впереди нет ни одного свободного места (жена проснулась в тот момент, когда автомобиль проезжал k-е место парковки, а все места $k, k+1, \ldots, m$ были заняты), законопослушность водителя перевешивает послушность жене и он, принеся свои извинения, едет дальше.

Предположим, что на одной улице оказалось n автомобилей с дремлющими женами, причем j-я жена (имеется в виду, конечно же, жена j-го водителя. — *Прим. перев.*) просыпается напротив a_j-го места парковки. Сколько последовательностей $a_1 \ldots a_n$ позволят всем автомобилям успешно припарковаться, если предположить, что первоначально улица была пуста и что ни один из припарковавшихся автомобилей не уехал. Например, при $m = n = 9$ и $a_1 \ldots a_9 = 3\ 1\ 4\ 1\ 5\ 9\ 2\ 6\ 5$ автомобили будут припаркованы следующим образом.



[*Указание.* Воспользуйтесь анализом линейного исследования.]

30. [*M38*] Покажите, что в задаче о парковке из упр. 29 при $n = m$ все машины припарковываются тогда и только тогда, когда существует перестановка $p_1 p_2 \ldots p_n$ множества $\{1, 2, \ldots, n\}$, такая, что $a_j \le p_j$ для всех j.

31. [*M40*] При $n = m$ в упр. 29 количество решений равно $(n+1)^{n-1}$, а из упр. 2.3.4.4–22 известно, что это — число свободных деревьев на $n+1$ помеченной вершине! Найдите интересную связь между последовательностью парковок и деревьями.

32. [*M27*] Докажите, что система уравнений (44) имеет единственное решение, $(c_0, c_1, \ldots, c_{M-1})$, при любых целых неотрицательных $b_0, b_1, \ldots, b_{M-1}$, сумма которых меньше M. Разработайте алгоритм для поиска этого решения.

▶ **33.** [*M23*] Объясните, почему (51) представляет собой только оценку истинного среднего количества проб, выполняемых алгоритмом L. Что именно при выводе (51) привело к тому, что формула не абсолютно точна?

▶ **34.** [*M23*] Назначение этого упражнения заключается в исследовании среднего количества проб в хеш-таблице с цепочками, когда списки остаются раздельными, как на рис. 38.

a) Чему равна P_{Nk}, вероятность того, что данный список имеет длину k, если все M^N хеш-последовательности (35) равновероятны?

b) Найдите производящую функцию $P_n(z) = \sum_{k \ge 0} P_{Nk} z^k$.

c) Выразите среднее количество проб для успешного поиска через эту производящую функцию.

---

* Цитаты из произведений В. Высоцкого и В. Токарева, присутствующие в переводе данного и следующего упражнений, в оригинальном издании, конечно же, отсутствуют. — *Прим. перев.*

d) Выведите среднее количество проб в случае *неудачного* поиска, рассматривая варианты структур данных, в которых используются следующие соглашения: (i) хеш-адрес указывает заголовок списка (см. рис. 38); (ii) хеш-адрес указывает позицию в таблице (см. рис. 40), но все ключи, кроме первого в списке, попадают в отдельную область переполнения; (iii) хеш-адрес указывает позицию в таблице, и все элементы находятся в хеш-таблице.

35. [*M24*] Продолжая упр. 34, найдите, чему равно среднее число проб при неудачном поиске, когда отдельные списки упорядочены по значениям ключей? Рассмотрите структуры данных (i), (ii) и (iii).

36. [*M23*] Продолжая упр. 34, (d), найдите *дисперсию* числа проб при неудачном поиске с использованием структур данных (i) и (ii).

▶ **37.** [*M29*] Формула (19) дает среднее количество проб в случае успешного поиска по раздельным цепочкам. Чему равна *дисперсия* полученного числа проб?

38. [*M32*] (*Хеширование с использованием деревьев.*) Способный программист может попытаться использовать вместо линейных списков бинарные деревья поиска в методе цепочек, тем самым комбинируя алгоритм 6.2.2T с хешированием. Проанализируйте среднее число проб, которые требуются для такого комбинированного алгоритма как в случае успешного, так и в случае неудачного поиска. [*Указание.* См. 5.2.1–(15).]

39. [*M28*] Пусть $c_N(k)$ — общее количество списков длиной k, сформированных при помощи алгоритма C, который применен ко всем M^N хеш-последовательностям (35). Найдите рекуррентное соотношение между числами $c_N(k)$, позволяющее определить простую формулу для суммы

$$S_N = \sum_k \binom{k}{2} c_N(k).$$

Каким образом S_N связано с числом проб при неудачном поиске по алгоритму C?

40. [*M33*] Формула (15) дает среднее количество проб, выполняемое алгоритмом C при неудачном поиске. Чему равна *дисперсия* этого числа проб?

41. [*M40*] Проанализируйте T_N, среднее количество уменьшения индекса R на 1 при вставке $(N+1)$-го элемента по алгоритму C.

▶ **42.** [*M20*] Выведите (17), вероятность того, что первая проба алгоритма C оказалась успешной.

43. [*HM44*] Проанализируйте модификацию алгоритма C, в которой используется таблица размером $M' \geq M$. Для хеширования используются только первые M позиций, так что первые $M' - M$ пустых узлов, найденных на шаге C5, будут находиться в дополнительных позициях таблицы. Какой в случае фиксированного значения M' выбор $1 \leq M \leq M'$ приведет к наивысшей производительности?

44. [*M43*] (*Случайные пробы с вторичной кластеризацией.*) Цель данного упражнения состоит в определении ожидаемого количества проб в схеме с открытой адресацией с последовательностью проб

$$h(K), \quad (h(K)+p_1) \bmod M, \quad (h(K)+p_2) \bmod M, \quad \ldots, \quad (h(K)+p_{M-1}) \bmod M,$$

где $p_1 p_2 \ldots p_{M-1}$ — случайно выбранная перестановка множества $\{1, 2, \ldots, M-1\}$, зависящая от $h(K)$. Другими словами, все ключи с одинаковыми значениями $h(K)$ следуют одной и той же последовательности опробований и все $(M-1)!^M$ возможных выборов M последовательностей проб равновероятны.

Эта ситуация может быть точно смоделирована с помощью следующей экспериментальной процедуры, выполняемой над первоначально пустым линейным массивом размером m. Выполните приведенные ниже операции n раз. "С вероятностью p займите

крайнюю слева пустую позицию. В противном случае (т. е. с вероятностью $q = 1 - p$) выберите любую позицию таблицы, за исключением крайней слева, причем все $m - 1$ позиций должны быть равновероятными. Если выбранная позиция пуста, займите ее; иначе выберите *любую* пустую позицию (включая крайнюю слева) и займите ее (все пустые позиции рассматриваются как равновероятные).

Например, при $m = 5$ и $n = 3$ конфигурационный массив (занято, занято, пусто, занято, пусто) после выполнения этой процедуры образуется с вероятностью

$$\tfrac{7}{192}qqq + \tfrac{1}{6}pqq + \tfrac{1}{6}qpq + \tfrac{11}{64}qqp + \tfrac{1}{3}ppq + \tfrac{1}{4}pqp + \tfrac{1}{4}qpp.$$

(Данная процедура соответствует случайному исследованию с вторичной кластеризацией при $p = 1/m$, поскольку можно перенумеровать элементы таблицы так, что некоторая последовательность проб будет равна 0, 1, 2, ..., в то время как все остальные окажутся случайными.)

Найдите формулу для среднего количества занятых позиций на левом конце массива (в приведенном примере — 2). Найдите также асимптотическое значение этой величины при $p = 1/m$, $n = \alpha(m + 1)$ и $m \to \infty$.

45. [*M43*] Выполните упр. 44, но с *третичной кластеризацией*, когда последовательность проб начинается с $h_1(K)$, $((h_1(K) + h_2(K)) \bmod M$, а дальнейшие пробы выбираются случайным образом в зависимости только от $h_1(K)$ и $h_2(K)$. (Таким образом, $(M-2)!^{M(M-1)}$ возможных выборов $M(M-1)$ последовательностей проб с этим свойством рассматриваются как равновероятные.) Является ли эта процедура асимптотическим эквивалентом равномерного исследования?

46. [*M42*] Определите C'_N и C_N для метода открытой адресации, использующего последовательность проб

$$h(K),\ 0,\ 1,\ \ldots,\ h(K) - 1,\ h(K) + 1,\ \ldots,\ M - 1.$$

47. [*M25*] Найдите среднее количество проб, необходимых при открытой адресации для последовательности проб

$$h(K),\ h(K) - 1,\ h(K) + 1,\ h(K) - 2,\ h(K) + 2,\ \ldots.$$

Эта последовательность проб была однажды предложена в связи с тем, что все расстояния между последовательными пробами различны при четном M. [*Указание.* Небольшой трюк — и задача становится очень простой.]

▸ **48.** [*M21*] Проанализируйте метод открытой адресации, позиции проб которого $h_1(K)$, $h_2(K)$, $h_3(K), \ldots$ представляют собой бесконечную последовательность взаимно независимых случайных хеш-функций $(h_n(K))$. В этом случае возможно опробование одной и той же позиции дважды, например, если $h_1(K) = h_2(K)$, но такое совпадение крайне редко, пока таблица не станет близка к заполненной.

49. [*HM24*] Определите среднее количество проб (обращений к внешней памяти) C_N и C'_N для цепочек с раздельными списками, предполагая, что список, содержащий k элементов, требует $\max(1, k - b + 1)$ проб при неудачном запуске. Вместо точной вероятности P_{Nk}, как в упр. 34, воспользуйтесь *приближением Пуассона*

$$\binom{N}{k}\left(\frac{1}{M}\right)^k\left(1 - \frac{1}{M}\right)^{N-k} = \frac{N}{M}\frac{N-1}{M}\cdots\frac{N-k+1}{M}\left(1 - \frac{1}{M}\right)^N\left(1 - \frac{1}{M}\right)^{-k}\frac{1}{k!}$$
$$= \frac{e^{-\rho}\rho^k}{k!}(1 + O(k^2/M)),$$

справедливым для $N = \rho M$ и $k \le \sqrt{M}$ при $M \to \infty$; выведите формулы (57) и (58).

50. [*M20*] Покажите, что $Q_1(M,N) = M - (M-N-1)Q_0(M,N)$ в обозначениях упр. 42. [*Указание.* Сначала докажите, что $Q_1(M,N) = (N+1)Q_0(M,N) - NQ_0(M,N-1)$.]

51. [*HM17*] Выразите функцию $R(\alpha,n)$, определенную в (55), через функцию Q_0, определенную в (42).

52. [*HM20*] Докажите, что $Q_0(M,N) = \int_0^\infty e^{-t}(1+t/M)^N\,dt$.

53. [*HM20*] Докажите, что функция $R(\alpha,n)$ может быть выражена через неполные гамма-функции, и используйте результат упр. 1.2.11.3–9 для нахождения асимптотического значения $R(\alpha,n)$ с точностью $O(n^{-2})$ при $n\to\infty$ для фиксированного $\alpha<1$.

54. [*HM28*] Покажите, что при $b=1$ формула (61) эквивалентна (23). *Указание.* Имеем

$$t_n(\alpha) = \frac{(-1)^{n-1}}{n!\,\alpha}\sum_{m>n}\frac{(-n\alpha)^m}{m(m-1)(m-n-1)!}.$$

55. [*HM43*] Обобщите модель Шея-Спрута, обсуждавшуюся после теоремы P, для M блоков размером b. Докажите, что $C(z)$ равно $Q(z)/(B(z)-z^b)$, где $Q(z)$ — полином степени b и $Q(1)=0$. Покажите, что среднее число проб составляет

$$1+\frac{M}{N}C'(1) = 1+\frac{1}{b}\left(\frac{1}{1-q_1}+\cdots+\frac{1}{1-q_{b-1}}-\frac{1}{2}\frac{B''(1)-b(b-1)}{B'(1)-b}\right),$$

где $q_1,\ldots,q_{b-1}$ — корни $Q(z)/(z-1)$. Заменив биномиальное распределение вероятностей $B(z)$ приближением Пуассона $P(z)=e^{b\alpha(z-1)}$, где $\alpha=N/Mb$, и использовав формулу обращения Лагранжа (см. 2.3.4.4–(9) и упр. 4.7–8), приведите ответ к виду (61).

56. [*HM43*] Обобщите теорему K, получив точный анализ линейного исследования при блоках размером b. Чему равно асимптотическое число проб в случае успешного поиска при заполненной таблице ($N=Mb$)?

57. [*M47*] Дает ли назначение одинаковых вероятностей последовательностям проб минимальное значение C_N среди всех методов открытой адресации?

58. [*M21*] (С. К. Джонсон (S. C. Johnson).) Найдите десять перестановок множества $\{0,1,2,3,4\}$, которые эквивалентны равномерному исследованию в смысле теоремы U.

59. [*M25*] Докажите, что если назначение вероятностей перестановкам эквивалентно равномерному исследованию в смысле теоремы U, то при достаточно большом M число перестановок с ненулевыми вероятностями превосходит M^a для любого фиксированного показателя степени a.

60. [*M47*] Будем говорить, что схема открытой адресации включает *единственное хеширование*, если в ней используется в точности M последовательностей проб, начинающихся со всех возможных значений $h(K)$, каждое из которых встречается с вероятностью $1/M$.

Является ли наилучшая схема единственного хеширования (в смысле минимума C_N) асимптотически лучше случайных схем, описанных в (29)? В частности, справедливо ли $C_{\alpha M}\ge 1+\frac{1}{2}\alpha+\frac{1}{2}\alpha^2+O(\alpha^3)$ при $M\to\infty$?

61. [*M46*] Является ли метод, анализировавшийся в упр. 46, наихудшей возможной схемой с единственным хешированием в смысле упр. 60?

62. [*M49*] Схема с единственным хешированием называется *циклической*, если увеличения $p_1p_2\ldots p_{M-1}$ (в обозначениях упр. 44) фиксированы при всех K. (Примерами такого метода являются линейное исследование и последовательности, рассмотренные в упр. 20 и 47.) *Оптимальной* схемой с единственным хешированием является та, значение C_M которой минимально среди всех $(M-1)!^M$ схем с единственным хешированием при данном M. При $M\le 5$ наилучшая схема с единственным хешированием является циклической. Справедливо ли это для любого M?

63. [*M25*] Если в хеш-таблице выполняются случайные вставки и удаления, то сколько независимых вставок в среднем следует сделать, чтобы все M позиций были занятыми в то или иное время? (Это значение представляет собой среднее время работы на отказ метода удаления, просто помечающего ячейки как "удаленные".)

64. [*M41*] Проанализируйте ожидаемое поведение алгоритма R (удаление с линейным исследованием). Сколько раз в среднем выполняется шаг R4?

▶ **65.** [*20*] (*Ключи переменной длины.*) Во многих приложениях с хеш-таблицами используются ключи, представляющие набор символов произвольной длины. В таких приложениях нельзя просто хранить ключ в таблице, как в программах из этого раздела. Каким образом можно организовать работу хеш-таблиц с ключами переменной длины на MIX-компьютере?

▶ **66.** [*25*] (Оле Амбль (Ole Amble), 1973.) Можно ли так вставить ключи в открытую хеш-таблицу с использованием их числового или алфавитного порядка, чтобы в случае поиска по алгоритму L или D можно было сделать вывод о его неудачном завершении, встретив ключ, *меньший*, чем аргумент поиска?

67. [*M41*] Пусть N ключей с хеш-адресами $a_1\, a_2 \ldots a_N$ вставляются в хеш-таблицу согласно алгоритму L и пусть d_j — смещение j-го ключа из его начального положения a_j; тогда $C_N = 1 + (d_1 + d_2 + \cdots + d_N)/N$. Теорема P гласит, что перестановка a не влияет на сумму $d_1 + d_2 + \cdots + d_N$; однако такая перестановка может значительно изменить сумму $d_1^2 + d_2^2 + \cdots + d_N^2$. Например, хеш-последовательность 1 2 ... $N{-}1$ $N{-}1$ делает $d_1\, d_2 \ldots d_{N-1}\, d_N$ равным 0 0 ... 0 $N{-}1$ и $\sum d_j^2 = (N-1)^2$, в то время как ее зеркальное отражение $N{-}1$ $N{-}1$ ... 2 1 приводит к более "цивилизованному" перемещению 0 1 ... 1 1, для которого $\sum d_j^2 = N - 1$.

a) Какое размещение $a_1\, a_2 \ldots a_N$ минимизирует $\sum d_j^2$?

b) Поясните, каким образом следует модифицировать алгоритм L, чтобы после каждой вставки сохранялся набор перемещений с наименьшей дисперсией.

c) Определите среднее значение $\sum d_j^2$ с указанной модификацией и без нее.

68. [*M41*] Чему равна дисперсия среднего числа проб в случае успешного поиска по алгоритму L? В частности, чему равно среднее $(d_1 + d_2 + \cdots + d_N)^2$ в обозначениях упр. 67?

69. [*M25*] (Эндрю Яо (Andrew Yao).) Докажите, что схемы циклического единственного хеширования удовлетворяют неравенству $C'_{\alpha M} \geq \frac{1}{2}(1 + 1/(1-\alpha))$ в смысле упр. 62. [*Указание.* Покажите, что для неудачного поиска требуется ровно k проб с вероятностью $p_k \leq (M-N)/M$.]

70. [*ḦM43*] Докажите, что ожидаемое количество проб, необходимых для вставки $(\alpha M + 1)$-го элемента при помощи двойного хеширования, не превышает ожидаемого количества проб, необходимых для вставки $(\alpha M + \sqrt{O(\log M)/M})$-го элемента при равномерном исследовании.

71. [*40*] Поэкспериментируйте с поведением алгоритма C при его адаптации к внешнему поиску так, как описано в тексте.

▶ **72.** [*M28*] (*Универсальное хеширование.*) Представьте гигантскую матрицу H, имеющую по одному столбцу для каждого возможного ключа K. Элементы H представляют собой числа от 0 до $M-1$; строки H представляют хеш-функции. Мы говорим, что H определяет *универсальное семейство хеш-функций*, если любые два столбца совпадают не более чем в R/M строках, где R — общее количество строк.

a) Докажите, что если H универсальна в этом смысле и хеш-функция h выбирается случайным образом из строки H, то ожидаемый размер списка, содержащего все данные ключи K в методе раздельных цепочек (см. рис. 38) после вставки любого множества различных ключей $K_1, K_2, \ldots, K_N$ будет равен $\leq 1 + N/M$.

b) Предположим, что каждый h_j в (9) представляет собой случайно выбранное отображение множества всех символов на множество $\{0, 1, \ldots, M-1\}$. Покажите, что это соответствует универсальному семейству хеш-функций.

c) Останется ли результат (b) справедливым, если $h_j(0) = 0$ для всех j, но $h_j(x)$ — случайно при $x \neq 0$?

73. [*M26*] (Картер (Carter) и Вегман (Wegman).) Покажите, что п. (b) предыдущего упражнения выполняется, даже когда h_j не являются полностью случайными функциями, но имеют один из следующих видов. (i) Пусть x_j — двоичное число $(b_{j(n-1)} \ldots b_{j1} b_{j0})_2$. Тогда

$$h_j(x_j) = (a_{j(n-1)} b_{j(n-1)} + \cdots + a_{j1} b_{j1} + a_{j0} b_{j0}) \bmod M,$$

где каждое a_{jk} выбирается случайно по модулю M. (ii) Пусть M — простое число; положим $0 \le x_j < M$. Тогда

$$h_j(x_j) = (a_j x_j + b_j) \bmod M,$$

где a_j и b_j выбираются случайным образом по модулю M.

74. [*M29*] Пусть H определяет универсальное семейство хеш-функций. Докажите или опровергните следующее. Даны N различных столбцов и некоторая случайным образом выбранная строка. При этом ожидаемое число нулей в этой строке, принадлежащих выбранным столбцам, равно $O(1) + O(N/M)$. [Таким образом, каждый список в методе раздельных цепочек будет иметь ожидаемый размер.]

75. [*M26*] Докажите или опровергните следующие утверждения о хеш-функции h из (9), если h_j представляют собой независимые случайные функции.

a) Вероятность того, что $h(K) = m$, равна $1/M$ для всех $0 \le m < M$.

b) Если $K \neq K'$, вероятность того, что $h(K) = m$ и $h(K') = m'$, равна $1/M^2$ для всех $0 \le m, m' < M$.

c) Если K, K' и K'' различны, вероятность того, что $h(K) = m$, $h(K') = m'$ и $h(K'') = m''$, равна $1/M^3$ для всех $0 \le m, m', m'' < M$.

d) Если K, K', K'' и K''' различны, вероятность того, что $h(K) = m$, $h(K') = m'$, $h(K'') = m''$ и $h(K''') = m'''$, равна $1/M^4$ для всех $0 \le m, m', m'', m''' < M$.

▶ **76.** [*M21*] Предложите путь изменения (9) для ключей с переменной длиной, сохраняющего свойства универсального хеширования.

77. [*M22*] Пусть H определяет универсальное семейство хеш-функций из 32-битовых ключей в 16-битовые (так что H имеет 2^{32} столбца, а в обозначениях упр. 72 $M = 2^{16}$). 256-битовый ключ можно рассматривать как конкатенацию восьми 32-битовых частей

$$x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8;$$

его можно отобразить в 16-битовый адрес при помощи хеш-функции

$$h_4(h_3(h_2(h_1(x_1)h_1(x_2))h_2(h_1(x_3)h_1(x_4)))h_3(h_2(h_1(x_5)h_1(x_6))h_2(h_1(x_7)h_1(x_8)))),$$

где h_1, h_2, h_3 и h_4 — случайно и независимо выбранные строки H (здесь, например, $h_1(x_1)h_1(x_2)$ означает 32-битовое число, полученное конкатенацией $h_1(x_1)$ и $h_1(x_2)$). Докажите, что вероятность того, что два различных ключа хешируются в один и тот же адрес, меньше 2^{-14}. [Для этой схемы требуется гораздо меньше случайных выборов, чем для (9).]

Она сделала из названий настоящий хаш, это уж точно.*

— ГРАНТ АЛЛЕН (GRANT ALLEN)

(Шатры Сима (The Tents of Shem), 1889)

"HASH, x". Для этого слова нет определения — никто не знает, что такое "hash".

— АМБРУАЗ БИРС (AMBROSE BIERCE)

(Дьявольский словарь (The Devil's Dictionary), 1906)

---

* В переводе пришлось использовать очень схожее по звучанию (и даже по смыслу) слово *хаш*, означающее армянское горячее блюдо из рубленого мяса. — *Прим. перев.*

6.5. ВЫБОРКА ПО ВТОРИЧНЫМ КЛЮЧАМ

Мы закончили изучение поиска по *первичным ключам*, т. е. по ключам, которые однозначно определяют запись в файле. Однако иногда необходимо выполнить поиск, основанный на значениях других полей записей, помимо первичного ключа. Эти другие поля часто именуются *вторичными ключами* или *атрибутами* записи. Например, имея файл регистрации с информацией о студентах университета, можно найти всех второкурсников из Огайо, не специализирующихся по математике или статистике, или всех незамужних франкоговорящих аспиранток...

В общем случае полагается, что в каждой записи содержится несколько атрибутов, и необходимо найти все записи с некоторыми значениями этих атрибутов. Определение требуемых записей называется *запросом* (*query*). Обычно запросы подразделяются на следующие три типа.

a) *Простой запрос*, определяющий конкретное значение некоторого атрибута, например "СПЕЦИАЛИЗАЦИЯ = МАТЕМАТИКА" или "МЕСТОЖИТЕЛЬСТВО.ШТАТ = ОГАЙО".

b) *Запрос диапазона*, запрашивающий определенный диапазон значений некоторого атрибута, например "ЦЕНА < $18.00" или "21 < ВОЗРАСТ ≤ 23".

c) *Логический запрос*, состоящий из запросов предыдущих типов, скомбинированных при помощи логических операций AND, OR, NOT, например

"(КУРС = ВТОРОКУРСНИК) AND (МЕСТОЖИТЕЛЬСТВО.ШТАТ = ОГАЙО)
AND NOT ((СПЕЦИАЛИЗАЦИЯ = МАТЕМАТИКА) OR (СПЕЦИАЛИЗАЦИЯ = СТАТИСТИКА))".

Проблема разработки эффективных технологий поиска достаточно сложна уже для этих трех простых типов запросов, а потому более сложные типы запросов обычно не рассматриваются. Например, железнодорожная компания может иметь файл с информацией о текущем состоянии всех товарных вагонов. Запрос наподобие "Найти все пустые вагоны-рефрижераторы в радиусе 500 миль от Сиэтла" при этом в явном виде невозможен, за исключением случая, когда "расстояние от Сиэтла" является атрибутом, хранящимся в каждой записи, а не сложной функцией вычисления этой величины по значениям других атрибутов. Использование же логических кванторов в дополнение к AND, OR, NOT приведет к дальнейшему усложнению запросов, ограниченных исключительно воображением запрашивающего. Имея файл со статистикой по бейсболу, например, можно запросить информацию о самой длинной серии удачных ударов в ночных играх. Такие запросы несмотря на их сложность могут выполняться в течение одного прохода по соответствующим образом упорядоченному файлу. Однако могут быть и существенно более сложные запросы, например "Найти все пары записей, имеющие одинаковые значения в пяти и более атрибутах (без указания атрибутов, которые должны совпадать)". Такие запросы могут рассматриваться как общие задачи программирования, выходящие за рамки нашего обсуждения, хотя зачастую они могут быть разбиты на подзадачи, подобные рассматриваемым здесь.

Прежде чем приступить к изучению различных технологий получения информации по вторичным ключам, стóит рассмотреть экономический аспект вопроса. Хотя огромное количество приложений помещается в жесткие рамки трех типов запросов, описанных выше, далеко не для всех из них подходят технологии, которые мы будем изучать; в некоторых из них лучше было бы использовать ручную, а не машинную работу! Люди взобрались на Эверест просто потому, что он существует.

И многое альпинистское снаряжение было разработано именно по этой причине. Так и в информационных технологиях: увидев перед собой Эверест данных, люди тут же начинают разрабатывать снаряжение для его покорения, позволяющее в оперативном режиме ответить на все вопросы, которые только могут присниться в кошмарном сне. Зачастую они просто не задумываются о вопросах стоимости. Такие вычисления возможны, но они не подходят для любого приложения.

Например, рассмотрим следующий простой подход к выборке по вторичным ключам: после сбора *пакета* из нескольких запросов (*batching*) выполним последовательный поиск по всему файлу, получая все нужные записи. (Сбор пакета означает, что мы накапливаем некоторое количество запросов перед их выполнением.) Такой метод вполне удовлетворителен при не слишком больших файлах и отсутствии требования немедленно обработать поступающие запросы. Этот подход может использоваться с файлами на лентах и требует работы компьютера над запросом через некоторые интервалы времени, что делает его вполне экономичным в плане стоимости оборудования. Более того, он способен обрабатывать вычислительные запросы наподобие рассмотренного нами ранее (с использованием понятия "расстояние от Сиэтла").

Другой простой путь облегчить получение информации по вторичному ключу состоит в делегировании части работы *человеку*, который снабжен подходящими печатными указателями. Этот метод зачастую является наиболее разумным и экономичным способом работы (если, конечно, старая бумага перерабатывается при издании нового указателя), в особенности потому, что люди обычно отмечают интересные запросы при обеспечении удобного доступа к данным большого объема*.

Приложения, для которых не подходят приведенные простые схемы, включают в себя очень большие файлы, критичные ко времени ответа на запрос. Такая ситуация, например, возникает при одновременной выдаче запросов от множества пользователей или при машинной генерации запросов. Назначение данного раздела — рассмотреть способы обеспечения выборки информации при различных предположениях о структуре файла. К счастью, методы, которые будут обсуждаться, становятся все более и более пригодными для практического применения, а цена вычислений продолжает резко снижаться.

Существует немало хороших способов решения поставленных задач, однако (как читатель должен был понять из всех предварительных примечаний) все эти алгоритмы не так хороши, как алгоритм поиска информации по первичному ключу. Из-за огромного разнообразия файлов и приложений мы не в состоянии сколь-нибудь пóлно обсудить все возможности или проанализировать поведение каждого алгоритма в типичных условиях. В оставшейся части этого раздела представлены основные предлагаемые подходы, и дело читателя — решать, какая комбинация описанных технологий лучше всего подходит для решения той или иной конкретной задачи.

---

* Пожалуй, автор несколько недооценивает тенденции развития вычислительной техники и технологий баз данных. Так, например, оператор службы "09", конечно же, держит в голове сотни телефонов и на самые часто задаваемые вопросы способен ответить мгновенно. Но кто мешает использовать, например, кэширование запросов, которое не подвержено той же усталости к концу смены? Вычислительная техника и информационные технологии — очень быстро развивающаяся область, и то, что еще вчера казалось недостижимым, сегодня становится обыденным. Общая тенденция такова, что думать и принимать решения — это работа человека, а механически искать информацию — удел компьютера. — *Прим. перев.*

Инвертированные файлы. Первый важный класс технологий поиска по вторичному ключу основан на идее *инвертированных файлов*. Данный термин не означает, что файл перевернут вверх ногами; это означает изменение ролей атрибутов записей (вместо списка атрибутов записи приводится список записей с данным атрибутом).

Мы часто сталкиваемся с инвертированными файлами (пусть и под другими названиями) в нашей повседневной жизни. Например, инверсией англо-русского словаря будет словарь русско-английский; инвертированным файлом, соответствующем какой-нибудь книге, является ее предметный или авторский указатель. Бухгалтеры традиционно используют в работе "двойную бухгалтерию" (double-entry bookkeeping), в которой все сделки фиксируются как в кассовых книгах, так и в счетах клиентов, чтобы иметь возможность быстро получить информацию как по кассе, так и по клиентам.

В общем случае инвертированный файл не самодостаточен и должен использоваться совместно с "прямым", неинвертированным файлом. В инвертированном файле содержится избыточная информация — цена, которую приходится платить за ускорение поиска по вторичному ключу. Компоненты инвертированного файла называются *инвертированными списками*, т. е. списками всех записей с данным значением некоторого атрибута.

Подобно прочим спискам, инвертированные списки могут быть представлены в компьютере самыми разными способами, причем для каждого приложения и данных может быть выбран свой, наиболее подходящий способ представления. Одни вторичные ключи могут иметь только два значения (например, графа анкеты ПОЛ (хотя, как говорят, находятся уникумы, пишущие в этой графе что-то вроде "паркетный", а в англоязычной анкете на вопрос SEX отвечающие "regular"; впрочем, эти замечания больше относятся к вопросу о проверке корректности вводимых данных. — *Прим. перев.*)), и соответствующие списки могут оказаться весьма длинными: другие же, наподобие НОМЕР ТЕЛЕФОНА, как правило, коротки и с малым количеством дубликатов.

Представим, что хранить информацию в телефонной книге нужно так, чтобы все записи могли быть получены на основе имени, телефонного номера или адреса абонента. Одно из решений — создать три различных файла, ориентированных на поиск по своему типу ключа. Вторая идея состоит в комбинировании всех трех файлов, например, в три хеш-таблицы, служащие в качестве заголовков списков для метода цепочек. В последней схеме каждая запись файла должна быть элементом трех списков и, таким образом, должна иметь три поля ссылок. Этот так называемый *многосписочный* (*multilist*) метод проиллюстрирован на рис. 13 раздела 2.2.6 и обсуждается ниже. Третья возможность состоит в комбинировании трех файлов в один суперфайл по аналогии с каталогом библиотеки, в котором карточки авторов, названий книг и их тем располагаются вместе в алфавитном порядке.

После рассмотрения формата, использованного в предметном указателе этой книги, возникают новые идеи по представлению инвертированных списков. Для полей вторичного ключа, когда имеется около пяти элементов на одно значение атрибута, можно просто создать короткий последовательный список позиций записей (по аналогии с номерами страниц в указателе книги), следующих за значением ключа. В случае кластеризации записей может оказаться удобным указание диапазонов позиций (как, например, указание диапазона страниц 559–582). Если

записи в файле часто перемещаются, то лучше вместо позиций записей использовать первичные ключи — при этом не требуется обновление при перемещении записей. Например, ссылки в Библии всегда даются на главу и стих; во многих книгах ссылки основываются на номерах параграфов, а не страниц.

Ни одна из рассмотренных идей не подходит для атрибута с двумя значениями, например ПОЛ. В таком случае, конечно, достаточно только одного инвертированного списка, так как все не мужчины являются женщинами и наоборот. Если каждое значение соответствует примерно половине элементов файла, инвертированный список может быть ужасно длинным, но существует возможность получить очень изящное решение на бинарном компьютере с помощью представления в виде битовой строки, в которой каждый бит определяет значение конкретной записи. Так, битовая строка 01001011101... может означать, что первая запись в файле относится к мужчине, вторая — к женщине, следующие две — к мужчинам и т. д.

Таких методов достаточно для обработки простых запросов по определенным значениям атрибутов. Немного расширив эти методы, можно работать с запросами диапазонов, но вместо хеширования следует использовать схемы поиска, основанные на сравнениях (см. раздел 6.2).

Для логических запросов наподобие "(СПЕЦИАЛИЗАЦИЯ = МАТЕМАТИКА) AND (МЕСТОЖИТЕЛЬСТВО.ШТАТ = ОГАЙО)" необходимо найти пересечение двух инвертированных списков. Это можно сделать несколькими путями; например, если оба списка упорядочены, два прохода (по каждому из них) дадут все общие элементы. С другой стороны, можно выбрать *более короткий* список и просмотреть каждую из его записей, проверяя, соответствуют ли запросу прочие атрибуты. Однако этот метод работает только с запросами типа AND, но не OR и не очень хорош для внешних файлов, поскольку требует множества обращений к записям, не удовлетворяющим критериям запроса.

Такое же рассмотрение вопроса показывает, что организация в виде множества списков, описанная ранее, неэффективна для логических запросов к внешним файлам в связи с выполнением большого количества ненужных обращений. Например, представим, что указатель этой книги организован с помощью множества списков: каждый его элемент указывает только на последнюю страницу, на которой встречается соответствующий термин. Соответственно для каждого термина на этой странице имеется ссылка на предыдущую страницу с тем же термином и т. д. В таком случае для поиска всего материала, соответствующего запросу наподобие "[Анализ алгоритмов] AND ([Внешняя сортировка] OR [Внешний поиск])", придется перелистать очень много страниц. С другой стороны, ту же задачу можно решить, взглянув на две страницы имеющегося указателя и выполнив несложные операции над инвертированными списками для получения минимального подмножества страниц, которое удовлетворяет запросу.

Когда инвертированный список представлен в виде битовой строки, выполнение логических комбинаций простых запросов не вызывает трудностей, так как компьютеры работают с битовыми строками с относительно высокой скоростью. Для смешанных запросов, одни атрибуты которых представлены в виде последовательных списков записей, а другие — в виде битовых строк, несложно конвертировать последовательные списки в битовые строки, а затем выполнить над ними необходимые логические операции.

Здесь может оказаться небесполезным следующий пример. Предположим, что имеется 1 000 000 записей по 40 символов и файл хранится на дисках MIXTEC (см. раздел 5.4.9). Файл сам по себе заполняет два дисковых модуля, а инвертированные списки, вероятно, займут несколько больший объем. На каждой дорожке содержится 5 000 символов = 30 000 бит, так что инвертированный список для определенного атрибута займет не более 34 дорожек (это максимальное число дорожек соответствует самому короткому возможному представлению в виде битовой строки). Предположим, что имеется весьма сложный запрос, представляющий логическую комбинацию из десяти инвертированных списков. В худшем случае придется считать 340 дорожек информации из инвертированного файла с общим временем чтения $340 \times 25\,\mathrm{ms} = 8.5\,\mathrm{s}$. Среднее время задержки будет равно приблизительно половине времени чтения, но при аккуратном программировании его можно исключить. Сохраняя первые дорожки каждого битового списка на одном цилиндре, вторые — на следующем и т. д., можно практически исключить время поиска дорожки на диске, и в результате ожидать, что максимальное время поиска по диску составит около $34 \times 26\,\mathrm{ms} \approx 0.9\,\mathrm{s}$ (или в два раза больше при использовании двух независимых дисков). Наконец, если запросу удовлетворяют q записей, потребуется около $q \times (60\,\mathrm{ms}$ (поиск по диску) $+ 12.5\,\mathrm{ms}$ (скрытая задержка) $+ 0.2\,\mathrm{ms}$ (чтение)) дополнительного времени, чтобы получить их для последующей обработки. Таким образом, грубая оптимистическая оценка общего ожидаемого времени для обработки этого довольно сложного запроса равна $(10 + .073q)\,\mathrm{s}$. Полученное число может быть сопоставлено с примерно 210 s, требующимися для чтения всего файла с максимальной скоростью при тех же предположениях без использования инвертированных списков*. Этот пример показывает, что оптимизация по пространству тесно связана с оптимизацией по времени при работе с дисками; время обработки инвертированных списков примерно соответствует времени, необходимому для их поиска на диске и чтения.

В приведенном обсуждении в большей или меньшей степени полагалось, что файл не растет и не уменьшается во время запроса к нему. Однако что делать, если требуются частые обновления? Во многих приложениях для этого достаточно просто собрать в один пакет некоторое количество запросов на обновление, которые выполняются в момент, когда нет обрабатываемых запросов. Если же, напротив, обновление данных имеет наивысший приоритет, привлекательным представляется использование B-деревьев (см. раздел 6.2.4). Весь набор инвертированных списков может быть представлен в виде одного гигантского B-дерева со специальным соглашением о том, что узлы ветвей содержат значения ключей, а листья — как ключи, так и указатели на записи. Кроме того, обновленные версии файла могут обрабатываться и другими методами, которые будут рассмотрены ниже.

Геометрические данные. Огромное количество приложений работает с точками, линиями и прочими фигурами в двух или более измерениях. Одним из первых подходов к запросам, ориентированным на расстояния, было "почтовое дерево"

---

* Просьба смотреть на *относительные* значения приводимых автором величин, абстрагируясь от *абсолютных* значений, которые не соответствуют современной технике. Так, без специальной оптимизации в многозадачной OS/2 с жестким диском IDE при одновременном выполнении других операций для чтения 1 000 000 записей размером по 40 байт на компьютере переводчика этого раздела потребовалось 2.3 s. — *Прим. перев.*

(post-office tree), предложенное в 1972 году Брюсом Мак-Наттом (Bruce McNutt). Предположим, например, что необходимо ответить на запрос "Какой город является ближайшим к точке x?", имея заданную точку x. Каждый узел дерева Мак-Натта соответствует городу y и "тестовому радиусу" r; левое поддерево узла соответствует всем городам z, последовательно введенным в эту часть дерева, таким, что расстояние от y до $z \leq r + \delta$, а правое поддерево точно такое же для расстояний $\geq r - \delta$. Здесь δ — данный допуск; города, находящиеся на расстоянии от $r - \delta$ до $r + \delta$ от y, должны входить в *оба* поддерева. Поиск в таком дереве позволяет определить все города на расстоянии не более δ от заданной точки (рис. 45).

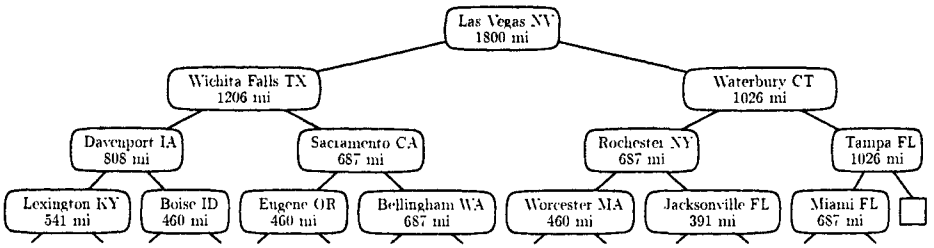


Рис. 45. Верхние уровни примера "почтового дерева". Для поиска всех городов вблизи данной точки x начинаем поиск от корня. Если x находится в пределах 1 800 миль от Лас-Вегаса, идем по дереву влево, в противном случае идем вправо. Повторяем этот процесс до тех пор, пока не будет достигнут конечный узел. Метод построения дерева гарантирует, что все деревья в пределах 20 миль от x встретятся в процессе поиска.

На основе этой идеи Мак-Натт и Эдвард Принг (Edward Pring) провели несколько экспериментов с использованием в качестве примера базы данных, содержащей 231 наиболее населенный город континентальной части США в случайном порядке. Тестовый радиус уменьшался регулярно, а именно заменой r на $0.67r$ при перемещении влево, и на $0.57r$ — при перемещении вправо, за исключением случая, когда последовательно выбирались две правые ветви (при этом r оставался неизменным). В итоге было получено дерево из 610 узлов при $\delta = 20$ миль и из 1 600 узлов при $\delta = 35$ миль. Верхние уровни меньшего из получившихся деревьев показаны на рис. 45. (В оставшейся части дерева Орландо (штат Флорида) появляется ниже Джексонвилля и Майями. Некоторые города встречаются очень часто. Так, Брактон (штат Массачусетс) встречается в 17 узлах!)

Быстрый рост файла при увеличении δ указывает на ограниченность применения почтовых деревьев. Пожалуй, лучше работать непосредственно с *координатами* каждой точки, рассматривая координаты как атрибуты или вторичные ключи; в таком случае можно создать логический запрос, основанный на диапазонах значений ключей. Например, предположим, что записи в файле относятся к городам Северной Америки и что запрос выполняет поиск всех городов, для которых справедливо

$$(21.49^\circ \leq \texttt{ШИРОТА} \leq 37.41^\circ) \quad \texttt{AND} \quad (70.34^\circ \leq \texttt{ДОЛГОТА} \leq 75.72^\circ).$$

Посмотрев на карту, можно увидеть, что множество городов удовлетворяет диапазону параметра ШИРОТА. Также найдется немало городов, имеющих соответствующий

параметр ДОЛГОТА, но сложно найти город, отвечающий обоим условиям одновременно. Один из подходов к подобным *ортогональным запросам диапазонов* состоит в разбиении множества всех возможных значений ШИРОТА и ДОЛГОТА таким образом, чтобы на каждый атрибут приходилось небольшое число классов (например, усечением значений до ближайшего меньшего значения, кратного 5°), и построении инвертированного списка по всем комбинированным классам (ШИРОТА, ДОЛГОТА). Это чем-то напоминает карты, состоящие из страниц для каждого локального региона. Используя 5-градусные интервалы, рассматриваемый здесь запрос будет обращаться к восьми страницам, а именно — $(20°, 70°)$, $(25°, 70°)$, ..., $(35°, 75°)$. Запрос диапазона в данном случае должен обработать каждую страницу, либо переходя к более мелким частям страницы, либо обращаясь непосредственно к записям — в зависимости от количества записей, соответствующих этой странице. По сути, получается структура дерева с двумерным ветвлением в каждом внутреннем узле.

Существенное усовершенствование этого подхода, называемое *сеточным файлом*, было разработано Ю. Нивергельтом (J. Nievergelt), Г. Гинтербергером (H. Hinterberger) и К. К. Севчик (K. C. Sevcik) [*ACM Trans. Database Systems* **9** (1984), 38–71]. Если каждая точка x имеет k координат $(x_1, \ldots, x_k)$, они разделяют значения i-й координаты на диапазоны

$$-\infty = g_{i0} < g_{i1} < \cdots < g_{ir_i} = +\infty \tag{1}$$

и положение x определяется индексами $(j_1, \ldots, j_k)$, такими, что

$$0 \le j_i < r_i, \qquad g_{ij_i} \le x_i < g_{i(j_i+1)} \qquad \text{для } 1 \le i \le k. \tag{2}$$

Совокупности точек, имеющих данное значение $(j_1, \ldots, j_k)$, называются *ячейками*. Записи для точек в одной и той же ячейке хранятся в одном блоке внешней памяти. Блоки могут также содержать точки из нескольких смежных ячеек, обеспечивая тем самым соответствие каждого блока k-мерному прямоугольному региону, или "суперячейке". Возможно использование различных стратегий для обновления значений границ сетки g_{ij} и для разделения и комбинирования блоков (см., например, K. Hinrichs, *BIT* **25** (1985), 569–592). Характеристики сеточных файлов со случайными данными проанализированы в работах M. Regnier, *BIT* **25** (1985), 335–357; P. Flajolet and C. Puech, *JACM* **33** (1986), 371–407, §4.2.

При простом способе работы с ортогональными запросами диапазонов, предложенном Дж. Л. Бентли (J. L. Bentley) и Р. А. Финкелем (R. A. Finkel), используются структуры, которые называются *четревьями** (*quadtrees*) [*Acta Informatica* **4** (1974), 1–9]. В двумерном случае каждый узел такого дерева представляет прямоугольник и содержит одну из точек в этом прямоугольнике. Имеется четыре поддерева, соответствующих четырем квадрантам исходного прямоугольника относительно координат данной точки. Аналогично для трех измерений существует восьмипутевое ветвление, и деревья соответственно называются *восьмиревьями* (*octrees*). k-мерные четревья, естественно, порождаются ветвлением по 2^k путям.

Математический анализ случайных четревьев является весьма сложным, однако в 1988 году независимо двумя группами исследователей (L. Devroye and L. Laforest,

* Переводя такие термины, поневоле чувствуешь себя Алисой в Зазеркалье, слушающей объяснение Шалтая-Болтая о словах, похожих на бумажник: открываешь его, а там два отделения. Надеюсь, читатель простит не слишком научно звучащие, зато оживляющие сухой текст термины. — *Прим. перев.*

SICOMP **19** (1990), 821–832; P. Flajolet, G. Gonnet, C. Puech, and J. M. Robson, *Algorithmica* **10** (1993), 473–500) была определена асимптотическая форма ожидаемого времени вставки N-го узла в случайное k-мерное четрево:

$$\frac{2}{k}\ln N + O(1). \tag{3}$$

Обратите внимание, что при $k = 1$ этот результат согласуется с хорошо известной формулой для вставки в бинарное дерево поиска 6.2.2–(5). Дальнейшие разработки Ф. Флажоле (P. Flajolet), Ж. Лабелля (G. Labelle), Л. Лафоре (L. Laforest) и Б. Салви (B. Salvy) показали, что в действительности средняя внутренняя длина пути может быть выражена в удивительно элегантной форме:

$$\sum_{l\geq 2}\binom{N}{l}(-1)^l\prod_{j=3}^{l}\Big(1-\frac{2^k}{j^k}\Big). \tag{4}$$

Таким образом, дальнейший анализ случайных четревьев возможен при помощи гипергеометрических функций [см. *Random Structures and Algorithms* **7** (1995), 117–144].

Бентли (Bentley) еще больше упростил представления четревьев, введя "k-d-деревья" (k-d trees), которые имеют только двойное ветвление в каждом узле [*CACM* **18** (1975), 509–517; *IEEE Transactions* **SE-5** (1979), 333–340]. 1-d-дерево представляет собой обычное бинарное дерево поиска, рассмотренное в разделе 6.2.2. 2-d-дерево подобно ему, но при ветвлении на четных уровнях сравниваются координаты x, а на нечетных — y. В общем случае k-d-дерево имеет узлы с k координатами и ветвление на каждом уровне базируется на сравнении одной из координат. Например, на уровне l происходит ветвление по координате $(k \bmod l) + 1$. Для гарантии того, что никакие две записи не будут иметь никаких совпадающих координат, можно использовать правило разрыва узлов (tie-breaking), основываясь на номерах записей или их положении в памяти компьютера. Случайно растущее k-d-дерево будет иметь то же значение средней длины пути и то же распределение ветвей, что и обычное бинарное дерево поиска, потому что предположения, лежащие в основе их роста, те же, что и в одномерном случае (см. упр. 6.2.2–6).

Если файл не изменяется динамически, можно сбалансировать любое k-d-дерево с N узлами так, чтобы его высота составляла $\approx \lg N$, выбрав среднее значение для ветвления в каждом узле. После этого можно быть уверенным в эффективности обработки запросов различных фундаментальных типов. Например, Бентли доказал, что можно найти все записи, имеющие t определенных координат, за $O(N^{1-t/k})$ шагов. Кроме того, можно найти все записи, лежащие в заданной прямоугольной области, не более чем за $O(tN^{1-1/k} + q)$ шагов, если всего имеется q таких записей, а t координат лежат в некоторых подобластях [D. T. Lee and C. K. Wong, *Acta Informatica* **23** (1977), 23–29]. В действительности, если данная область близка к кубической и q мало́ и если выбранные для ветвления координаты в каждом узле имеют наибольший разброс значений атрибутов, то, как показано в работе Friedman, Bentley, and Finkel, *ACM Trans. Math. Software* **3** (1977), 209–226, среднее время обработки запроса к такой области будет составлять всего лишь $O(\log N + q)$. Эта же формула применима при поиске в таком k-d-дереве ближайших соседей данной точки в k-мерном пространстве.

При использовании случайных, а не идеально сбалансированных k-d-деревьев среднее время работы для частичных совпадений по t определенным координатам несколько увеличивается до $\Theta(N^{1-t/k+f(t/k)})$. Функция f неявно определяется уравнением

$$\big(f(x)+3-x\big)^x\,\big(f(x)+2-x\big)^{1-x}=2 \tag{5}$$

и весьма мала: имеем

$$0 \le f(x) < 0.06329\,33881\,23738\,85718\,14011\,27797\,33590\,58170-. \tag{6}$$

При этом максимум достигается при x, близком к 0.585371 [см. P. Flajolet and C. Puech, *JACM* **33** (1986), 371–407, §3].

Рост популярности геометрических алгоритмов (и их эстетическая привлекательность) вызвали ускоренное развитие технологий решения многомерных задач и смежных вопросов разных видов. Фактически в 70-х годах появилось новое направление в математике и информатике, именуемое *вычислительной геометрией*. Отличным справочным пособием, в котором подробно излагается текущее состояние дел в этой отрасли знаний, является *Handbook of Discrete and Computational Geometry* под редакцией J. E. Goodman and J. O'Rourke (Boca Raton, Florida: CRC Press, 1997).

Всесторонний обзор структур данных и алгоритмов для важных случаев двух- и трехмерных объектов можно найти в двух взаимодополняющих книгах Ханана Самета (Hanan Samet) *The Design and Analysis of Spatial Data Structures* и *Applications of Spatial Data Structures* (Addison-Wesley, 1990). Самет обратил внимание на то, что исходные четревья Бентли и Финкеля более корректно было бы именовать "точечными четревьями" (point quadtrees). Название *четревья* теперь стало общим термином для любой иерархической декомпозиции геометрических данных.

Составные атрибуты. Два или более атрибутов могут быть скомбинированы в один суператрибут. Например, атрибут "(КУРС, СПЕЦИАЛИЗАЦИЯ)" может быть создан путем комбинирования полей КУРС и СПЕЦИАЛИЗАЦИЯ в университетском файле регистрации. Таким образом, запрос зачастую можно выполнить, объединяя короткие списки вместо пересечения длинных.

Идея комбинирования атрибутов была развита В. Ю. Лумом (V. Y. Lum) [*CACM* **13** (1970), 660–665], который предложил упорядочение инвертированных списков комбинированных атрибутов в лексикографическом порядке слева направо и создание нескольких копий с перестановкой индивидуальных атрибутов надлежащим способом. Например, предположим, что имеется три атрибута — A, B и C. Можно сформировать составные атрибуты

$$(\mathtt{A},\mathtt{B},\mathtt{C}),\qquad(\mathtt{B},\mathtt{C},\mathtt{A}),\qquad(\mathtt{C},\mathtt{A},\mathtt{B}) \tag{7}$$

и построить упорядоченные инвертированные списки для каждого из них. (Так, в первом списке записи упорядочены по их значениям A; записи с одинаковыми значениями A упорядочены по значениям B, а затем — по C.) Такая организация позволяет выполнять запросы, основанные на комбинации этих трех атрибутов; например, все записи, имеющие определенные значения A и C, будут располагаться в третьем списке последовательно.

Аналогично из атрибутов A, B, C и D можно сформировать шесть составных атрибутов:

$$(\mathtt{A},\mathtt{B},\mathtt{C},\mathtt{D}),\ (\mathtt{B},\mathtt{C},\mathtt{D},\mathtt{A}),\ (\mathtt{B},\mathtt{D},\mathtt{A},\mathtt{C}),\ (\mathtt{C},\mathtt{A},\mathtt{D},\mathtt{B}),\ (\mathtt{C},\mathtt{D},\mathtt{A},\mathtt{B}),\ (\mathtt{D},\mathtt{A},\mathtt{B},\mathtt{C}). \tag{8}$$

Они позволяют выполнять все комбинации простых запросов с фиксированными значениями одного, двух, трех или четырех атрибутов. Существует общая процедура построения $\binom{n}{k}$ комбинированных атрибутов из n отдельных атрибутов при $k \le \frac{1}{2}n$, такая, что все записи, имеющие определенные комбинации не более чем из k или не менее $n-k$ значений атрибутов, будут последовательно расположены в одном из списков комбинированных атрибутов (см. упр. 1). Можно обойтись и меньшим количеством комбинаций, если некоторые атрибуты имеют ограниченное множество значений. Например, если D представляет собой атрибут с двумя возможными значениями, то комбинации

$$(\mathtt{D},\mathtt{A},\mathtt{B},\mathtt{C}),\quad (\mathtt{D},\mathtt{B},\mathtt{C},\mathtt{A}),\quad (\mathtt{D},\mathtt{C},\mathtt{A},\mathtt{B}), \tag{9}$$

полученные в результате помещения D в (7), будут так же хороши, как и (8), с половинной избыточностью, поскольку запросы, не зависящие от D, могут быть обработаны путем просмотра в двух местах одного из списков.

Бинарные атрибуты. Поучительно рассмотреть частный случай, когда все атрибуты могут иметь только два значения. По сути, это *противоположность* комбинированных атрибутов, поскольку любое значение можно представить как двоичное число и рассматривать индивидуальные биты этого числа как отдельные атрибуты. В табл. 1 показан типичный файл с атрибутами "Да"–"Нет". В этом примере записи содержат рецепты домашнего печенья, а атрибуты определяют используемые ингредиенты. Например, миндальные вафли с ромом сделаны из масла, муки, молока, орехов и сахарного песка. Если рассматривать табл. 1 как матрицу из нулей и единиц, то транспонированная матрица будет представлять собой инвертированный файл в виде битовых строк.

Правый столбец табл. 1 используется для указания специальных, редко используемых продуктов. Они могут быть закодированы более эффективно, без отдельного столбца для каждого из них. То же самое справедливо для столбца "Кукурузный крахмал". Кроме того, можно найти более эффективный путь кодирования столбца "Мука", поскольку мука встречается во всех рецептах, кроме рецепта приготовления меренги. Однако пока оставим этот вопрос и просто проигнорируем столбец "Специальные ингредиенты".

Будем определять *базовый запрос* к файлу бинарных атрибутов как запрос на все записи, в одних столбцах которых содержатся нули, в других — единицы и в остальных столбцах — произвольные величины. Используя символ "*" для обозначения любого значения, базовый запрос можно представить в виде последовательностей символов 0, 1 и "*". Например, представим человека, который захотел печенья с кокосом и у которого аллергия на шоколад, который ненавидит анис и у которого дома закончился ванилин. Тогда его запрос может быть сформулирован так:

$$*\,0\,*\,*\,*\,*\,0\,*\,*\,1\,*\,*\,*\,*\,*\,*\,*\,*\,*\,*\,*\,*\,*\,*\,*\,*\,*\,0. \tag{10}$$

Из табл. 1 становится понятно, что его желания совпадают с его возможностями в случае ароматных палочек с черносливом.

Таблица 1

ФАЙЛ С БИНАРНЫМИ АТРИБУТАМИ

	Душистый перец	Зерна аниса	Пекарный порошок	Пищевая сода	Масло	Кардамон	Шоколад	Корица	Гвоздика	Кокосовый орех	Кофе	Кукурузный крахмал	Фники	Яичный белок	Яичный желток	Мука	Имбирь	Лимонный сок	Лимонная цедра	Молоко	Патока	Мускатный орех	Орехи	Толокно	Изюм	Соль	Сахар, жженый	Сахар, песок	Сахар, пудра	Ванилин	Специальные ингредиенты
Миндальные вафли с ромом	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	1	0	0	0	0	1	0	0	—
Печенье с яблочным соусом	0	0	0	1	1	0	0	1	1	0	0	0	0	1	1	1	0	0	0	0	0	1	1	0	1	0	0	1	1	1	Яблочный соус
Бананово-овсяное печенье	0	0	0	1	1	0	0	1	0	0	0	0	0	1	1	1	0	0	0	0	0	1	1	1	0	1	0	1	0	1	Бананы
Шоколадный хворост	0	0	0	1	1	0	1	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	1	0	0	1	0	1	0	1	—
Миндальное печенье с кокосами	0	0	1	0	1	0	0	0	0	1	0	0	0	1	1	1	0	0	0	0	0	0	1	0	0	1	0	1	0	1	—
Печенье со сливочным сыром	0	0	0	0	1	0	1	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	1	0	1	Сливочный сыр
Ароматные палочки с черносливом	0	0	0	0	1	0	0	0	0	1	0	1	0	1	1	1	0	0	0	0	0	0	1	0	0	1	1	0	0	0	Апельсины, чернослив
Драже в шоколаде	0	0	0	1	1	0	1	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	1	0	0	1	0	1	0	0	—
Райские палочки	0	0	1	0	1	0	0	0	0	1	0	0	0	1	1	1	0	0	0	0	0	0	1	0	0	1	1	0	0	1	—
Пирог с начинкой	0	0	1	0	1	0	0	0	0	0	0	0	1	1	1	1	0	1	1	0	0	0	1	0	0	1	0	1	0	1	—
Финский какор	0	0	0	0	1	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	1	0	0	1	0	1	0	0	Экстракт миндаля
Глазированные имбирные пряники	0	0	1	1	1	0	0	1	1	0	0	0	0	1	1	1	1	0	0	1	1	0	0	0	0	0	1	0	1	0	Уксус
Печенье с орехами	0	0	0	1	1	0	0	1	0	0	1	0	0	1	1	1	0	0	0	0	0	1	0	0	1	1	1	0	0	0	Абрикос
Драгоценное печенье	0	0	0	0	1	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	1	0	0	0	1	0	0	1	Смородиновое желе
Перепутаница	0	0	0	0	1	0	0	1	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	1	0	1	0	1	Растительное масло
Малайский крендель	0	0	0	0	1	1	0	0	0	0	0	0	0	1	1	1	0	0	1	0	0	0	1	0	0	0	0	1	0	0	—
Медовые пряники	1	0	0	1	0	0	0	1	1	0	0	0	0	1	1	1	0	1	1	1	0	1	1	0	0	1	1	0	1	0	Мед
Меренги	0	0	0	0	1	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	Засахаренная вишня
Моравское печенье со специями	1	0	0	1	1	0	0	1	1	0	0	0	0	1	0	1	1	0	0	0	1	1	0	0	0	1	1	0	1	0	—
Овсяные палочки с финиками	0	0	0	1	1	0	0	0	0	0	0	0	1	0	0	1	0	1	0	0	0	0	1	1	0	1	1	1	0	0	—
Старинное сахарное печенье	0	0	1	1	1	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	1	0	0	0	1	0	1	0	1	Сметана
Печенье с арахисовым маслом	0	0	0	1	1	0	1	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	1	1	0	0	1	Арахисовое масло
Юбочки	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	1	1	—
Печенье с перцем	1	1	1	0	0	1	0	1	1	0	1	0	0	1	1	1	0	0	1	0	0	1	1	0	0	1	1	0	1	0	Цукаты, перец, мускат
Шотландские овсяные коржики	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	1	0	1	0	1	—
Песочные звездочки	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	—
Анисовое печенье	0	1	1	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	1	0	0	0	0	0	0	1	0	1	1	0	—
Воздушное печенье	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	1	0	1	0	1	—
Шведский крендель	0	0	0	0	1	0	0	0	0	0	0	0	0	1	1	1	0	0	0	1	0	0	0	0	0	0	0	1	0	0	—
Швейцарское рассыпчатое печенье с корицей	0	0	0	0	1	0	0	1	0	0	0	0	0	1	1	1	0	0	0	1	0	0	0	0	0	1	1	1	0	0	—
Ириски	0	0	0	0	1	0	1	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	1	0	0	0	1	0	0	1	—
Ванильно-ореховое мороженое	0	0	1	0	1	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	1	0	0	1	0	1	0	1	—

McCall's Cook Book (New York: Random House, 1963), Chapter 9.

Прежде чем приступить к организации файла для базовых запросов, рассмотрим один важный специальный момент, когда в запросе отсутствуют нули, а есть только 1 и "*". Такой запрос можно назвать *включающим* (*inclusive query*), поскольку он запрашивает записи, которые *включают* некоторое множество атрибутов (если предположить, что 1 означает наличие определенного атрибута, а 0 — его отсутствие). Например, в табл. 1 два рецепта одновременно содержат и пекарный порошок, и пищевую соду — глазированные имбирные пряники и старинное сахарное печенье.

В некоторых приложениях достаточно использовать специальные включающие запросы, например в ручных карточных системах наподобие карт с перфорацией по краям или карт свойств. Система карт с перфорацией по краям для табл. 1 будет иметь по одной карточке на каждый рецепт с вырезами, соответствующими каждому ингредиенту (рис. 46). Обработка включающего запроса сводится к складыванию карточек аккуратной стопкой и введению спиц в положениях, соответствующих конкретным ингредиентам. После ввода спиц все карточки с определенными атрибутами выпадают из стопки.

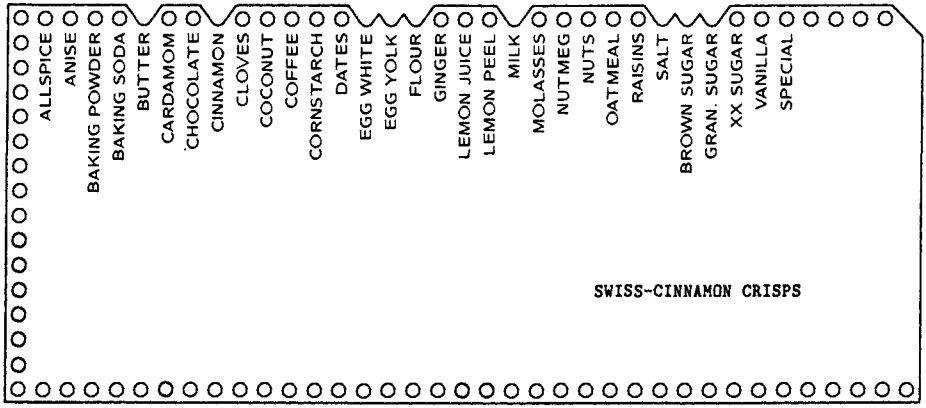


Рис. 46. Карта с перфорацией по краям.

Система, основанная на картах свойств, работает с инвертированным файлом аналогичным образом: имеется по одной карте для каждого атрибута и отверстия в карте пробиваются в позициях, которые соответствуют записям, содержащим этот атрибут. Таким образом, одна стандартная перфокарта шириной 80 позиций может использоваться для хранения информации о том, какие из $12 \times 80 = 960$ записей имеют данный атрибут. Для обработки включающего запроса отбираются карты свойств для определенных атрибутов и накладываются одна на другую. Луч света, проходя через позиции, в которых на всех картах имеются отверстия, покажет искомый результат. Это напоминает обработку логического запроса путем пересечения инвертированных битовых строк, как объяснялось выше.

Кодирование методом наложения. Причина, по которой нас (вооруженных современными компьютерами) интересуют методы поиска по карточкам вручную, заключается в том, что в свое время было разработано много хитроумных способов

хранения места на картах с перфорацией по краям, принципы которых применимы и для представления компьютерных файлов. Кодирование методом наложения представляет собой технологию, схожую с хешированием, хотя оно было разработано за несколько лет до изобретения хеширования. Идея заключается в отображении атрибутов в случайные k-битовые коды в n-битовых полях и наложении кодов каждого атрибута, имеющегося в записи. Включающий запрос для некоторого множества атрибутов может быть конвертирован во включающий запрос для соответствующих наложенных битовых кодов. Такому запросу могут удовлетворять несколько дополнительных записей, но количество подобных "ложных выпадений" может быть статистически учтено [см. Calvin N. Mooers, *Amer. Chem. Soc. Meeting* **112** (September, 1947), 14E–15E; *American Documentation* **2** (1951), 20–32].

В качестве примера кодирования методом наложения вновь обратимся к табл. 1, но только к той ее части, которая относится к специям, не рассматривая основные компоненты наподобие пекарного порошка, яиц и муки. В табл. 2 показано, что произойдет, если назначить случайные двухбитовые коды в десятибитовых полях каждой из специй и использовать наложение. Например, "Шоколадный хворост" будет получен в результате наложения кодов шоколада, орехов и ванилина:

$$0010001000 \vee 0000100100 \vee 0000001001 = 0010101101.$$

Наложение данных кодов даст также несколько ложных атрибутов; в нашем конкретном случае — душистый перец, засахаренную вишню, смородиновое желе, арахисовое масло и перец. Это вызовет ложные выпадения при некоторых запросах (тем самым будет предложено разработать новый рецепт — "Ложновыпадающее печенье"! :-)).

Для табл. 2 кодирование методом наложения работает не так уж хорошо, поскольку эта таблица представляет всего лишь маленький пример с большим количеством атрибутов. В самом деле, "Печенье с яблочным соусом" будет выпадать при *каждом* запросе, поскольку оно получено наложением семи кодов, покрывающих все десять позиций. Еще хуже обстоят дела в случае рецепта печенья с перцем (впрочем, с точки зрения кодов и ложных выпадений совершенно все равно, как получена цепочка из десяти единиц: наложением семи или двенадцати кодов. — *Прим. перев.*), полученного в результате наложения двенадцати кодов. С другой стороны, иногда табл. 2 работает на удивление неплохо; например, дав запрос "Ванилин", мы получим только одно ложное выпадение — "Печенье с перцем".

Более подходящий пример кодирования методом наложения получается при наличии, скажем, 32-битового поля и набора из $\binom{32}{3} = 4\,960$ различных атрибутов, где каждая запись может иметь до шести атрибутов и каждый атрибут кодируется тремя из 32 бит. В этой ситуации в предположении, что каждая запись имеет шесть случайно выбранных атрибутов, вероятности ложных выпадений в случае включающего запроса составляют:

по одному атрибуту	0.07948358	
по двум атрибутам	0.00708659	
по трем атрибутам	0.00067094	
по четырем атрибутам	0.00006786	(11)
по пяти атрибутам	0.00000728	
по шести атрибутам	0.00000082	

Таблица 2

ПРИМЕР КОДИРОВАНИЯ МЕТОДОМ НАЛОЖЕНИЯ

Коды отдельных приправ

Экстракт миндаля	0100000001	Финики	1000000100
Душистый перец	0000100001	Имбирь	0000110000
Зерна аниса	0000011000	Мед	0000000011
Яблочный соус	0010010000	Лимонный сок	1000100000
Абрикос	1000010000	Лимонная цедра	0011000000
Бананы	0000100010	Мускатный цвет	0000010100
Засахаренная вишня	0000101000	Патока	1001000000
Кардамон	1000000001	Мускатный орех	0000010010
Шоколад	0010001000	Орехи	0000100100
Корица	1000000010	Апельсины	0100000100
Цукаты	0100000010	Арахисовое масло	0000000101
Гвоздика	0001100000	Перец	0010000100
Кокосовый орех	0001010000	Чернослив	0010000010
Кофе	0001000100	Изюм	0101000000
Смородиновое желе	0010000001	Ванилин	0000001001

Наложенные коды

Миндальные вафли с ромом	0000100100	Медовые пряники	1011110111
Печенье с яблочным соусом	1111111111	Меренги	1000101100
Бананово-овсяное печенье	1000111111	Моравское печенье со специями	1001110011
Шоколадный хворост	0010101101	Овсяные палочки с финиками	1000100100
Миндальное печенье с кокосами	0001111101	Старинное сахарное печенье	0000011011
Печенье со сливочным сыром	0010001001	Печенье с арахисовым маслом	0010001101
Ароматные палочки с черносливом	0111110110	Юбочки	0000001001
Драже в шоколаде	0010101100	Печенье с перцем	1111111111
Райские палочки	0001111101	Шотландские овсяные коржики	0000001001
Пирог с начинкой	1011101101	Песочные звездочки	0000000000
Финский какор	0100100101	Анисовое печенье	0011011000
Глазированные имбирные пряники	1001110010	Воздушное печенье	0000001001
Печенье с орехами	1101010110	Шведский крендель	0000000000
Драгоценное печенье	0010101101	Швейцарское рассыпчатое печенье с корицей	1000000010
Перепутаница	1000001011	Ириски	0010101101
Малайский крендель	1011100101	Ванильно-ореховое мороженое	0000101101

Таким образом, если имеется M записей, которые не удовлетворяют двухатрибутному запросу, то около $0.007M$ записей будут иметь наложенные коды, соответствующие всем битам этих двух атрибутов (все вероятности вычисляются в упр. 4). Общее количество битов, необходимых для представления инвертированного файла, составляет 32, умноженное на количество записей, что меньше половины количества битов, требуемых для описания всех атрибутов в исходном файле.

При аккуратном использовании неслучайных кодов можно избежать ложных выпадений, как показано в работе W. H. Kautz and R. C. Singleton, *IEEE Trans.* **IT-10** (1964), 363–377; одна из таких конструкций рассматривается в упр. 16.

Малкольм Ч. Харрисон (Malcolm C. Harrison) [*CACM* **14** (1971), 777–779] обнаружил, что кодирование методом наложения может использоваться для ускорения *поиска текста*. Пусть нужно найти все вхождения некоторой строки символов в тексте большого объема без построения обширных таблиц алгоритма 6.3Р. Предположим также, что текст разделен на строки, например, по 50 символов: $c_1c_2\ldots c_{50}$. Харрисон предложил кодировать каждую из 49 пар c_1c_2, c_2c_3, ..., $c_{49}c_{50}$ методом хеширования каждой из них в число, скажем, от 0 до 127, а затем подсчитать "ключ" строки $c_1c_2\ldots c_{50}$, который представляет собой строку из 128 бит $b_0b_1\ldots b_{127}$, где $b_i = 1$ тогда и только тогда, когда $h(c_jc_{j+1}) = i$ для некоторого j.

Пусть необходимо найти все вхождения слова NEEDLE (ИГОЛКА) в большом файле с именем HAYSTACK (СТОГ СЕНА). При этом мы просто просмотрим все строки, ключи которых содержат 1 в позициях h(NE), h(EE), h(ED), h(DL) и h(LE). Считая хеш-функцию случайной, вероятность того, что случайная строка содержит все эти биты в своем ключе, можно оценить как 0.00341 (см. упр. 4). Следовательно, пересечение пяти инвертированных списков битовых строк позволит быстро обнаружить все строки, содержащие NEEDLE (естественно, с некоторым количеством ложных выпадений).

Предположение случайности на самом деле мало применимо для текста, поскольку обычный текст весьма избыточен и пары символов в нем распределены весьма неравномерно. Например, может оказаться полезным опустить все пары c_jc_{j+1}, содержащие символ пробела, в связи с тем, что пробел встречается в тексте гораздо чаще прочих символов.

Другое интересное приложение кодирования методом наложения к задачам поиска было предложено Бартоном Блюмом (Burton Bloom) [*CACM* **13** (1970), 422–426]. Его метод на самом деле применим для выборки по *первичным* ключам, хотя более подходящее место для его обсуждения — этот раздел. Представьте себе приложение для поиска в большой базе данных, которое не требует каких-либо дальнейших действий, если поиск оказался неудачным. Например, нужно просто проверить чью-то кредитную карточку или номер паспорта и, если в файле нет записей о данной персоне, никаких дальнейших действий предпринимать не требуется. Аналогичный метод применим и при компьютерной верстке для правильных переносов слов. Например, у нас есть метод расстановки переносов, корректно работающий с большинством слов, но имеющий некоторое количество исключений на 50 000 слов; если слово не найдено в файле исключений, можно использовать простой алгоритм.

В такой ситуации можно хранить таблицу битов во внутренней памяти, чтобы отсутствие большинства ключей распознавалось *без* обращений к внешней памяти. Вот как это можно сделать: обозначим внутреннюю битовую таблицу через $b_0b_1\ldots b_{M-1}$, где M очень велико. Для каждого ключа K_j в файле вычислим k независимых хеш-функций $h_1(K_j),\ldots,h_k(K_j)$ и установим соответствующие k бит равными 1 (эти k значений не обязаны быть различными). Таким образом, $b_i = 1$ тогда и только тогда, когда $h_l(K_j) = i$ для некоторых j и l. Теперь, чтобы определить наличие аргумента поиска K во внешнем файле, сначала выясним, справедливо ли соотношение $b_{h_l(K)} = 1$ при $1 \le l \le k$. Если нет, то нет и необходимости обращаться к

внешней памяти, но если соотношение справедливо, то при корректном выборе k и M последовательный поиск, вероятно, найдет K. Вероятность ложного выпадения при N записях в файле равна примерно $(1-e^{-kN/M})^k$. По сути, метод Блума рассматривает весь файл как одну запись, в которой первичные ключи представлены как атрибуты, а кодирование методом наложения производится в огромном M-битовом поле.

Еще один вариант кодирования методом наложения был разработан Ричардом А. Густафсоном (Richard A. Gustafson) [Ph. D. thesis (Univ. South Carolina, 1969)]. Предположим, что есть N записей и что каждая из них имеет шесть атрибутов, выбранных из 10 000 возможных. Запись может представлять, например, техническую статью, а атрибуты — ключевые слова, описывающие эту статью. Пусть h — это хеш-функция, отображающая каждый атрибут в число между 0 и 15. Запись с атрибутами $a_1, a_2, \ldots, a_6$ Густафсон предлагает отображать в 16-битовое число $b_0 b_1 \ldots b_{15}$, где $b_i = 1$ тогда и только тогда, когда $h(a_j) = i$ для некоторого j. И далее, если только $k < 6$ бит b равны 1, то другие $6 - k$ единиц добавляются некоторым случайным методом (необязательно зависящим от самой записи). Дано $\binom{16}{6} = 8\,008$ 16-битовых кодов, в которых имеется ровно шесть единичных битов, и при определенной доле везения примерно $N/8008$ записей будут отображены на каждое значение. Можно хранить 8 008 списков записей, непосредственно вычисляя адрес, соответствующий $b_0 b_1 \ldots b_{15}$, с использованием подходящей формулы. В самом деле, если единицы встречаются в позициях $0 \le p_1 < p_2 < \cdots < p_6$, то функция

$$\binom{p_1}{1} + \binom{p_2}{2} + \cdots + \binom{p_6}{6}$$

конвертирует каждую строку $b_0 b_1 \ldots b_{15}$ в единственное число между 0 и 8087, как будет показано в упр. 1.2.6–56 и 2.2.6–7.

Теперь, чтобы найти все записи, имеющие три определенных атрибута A_1, A_2 и A_3, вычислим $h(A_1)$, $h(A_2)$, $h(A_3)$; предполагая, что все эти значения различны, нужно будет просмотреть только записи, хранящиеся в $\binom{13}{3} = 286$ списках, битовые коды $b_0 b_1 \ldots b_{15}$ которых содержат единицы в соответствующих трех позициях. Другими словами, только $286/8008 \approx 3.5\%$ записей должны будут проверяться при поиске.

Превосходное описание кодирования методом наложения вместе с приложением для работы с большой базой данных телефонных номеров можно найти в статье C. S. Roberts, *Proc. IEEE/* **67** (1979), 1624–1642. Приложение метода к программному обеспечению проверки орфографии обсуждается в работе J. K. Mullin and D. J. Margoliash, *Software Practice & Exper.* **20** (1990), 625–630.

Комбинаторное хеширование. Идея, лежащая в основе только что описанного метода Густафсона, заключается в поиске некоторого пути отображения записей на адреса в памяти, такого, что к определенному запросу имеет отношение лишь сравнительно небольшое количество адресов памяти. Однако этот метод применим только к включающим запросам в том случае, когда отдельные записи обладают небольшим количеством атрибутов. Другой тип отображения, предназначенный для обработки произвольных базовых запросов типа (10), в которых содержатся нули, единицы и звездочки, был разработан в 1971 году Рональдом Л. Ривестом [см. *SICOMP* **5** (1976), 19–50].

Предположим сначала, что необходимо создать словарь для составления кроссвордов, содержащий все шестибуквенные слова английского языка; типичный запрос ищет все слова вида, например, N**D*E и получает ответ {NEEDLE, NIDDLE, NODDLE, NOODLE, NUDDLE}. Можно решить эту задачу при помощи 2^{12} списков, поместив слово NEEDLE в список номер

$$h(\mathtt{N})\, h(\mathtt{E})\, h(\mathtt{E})\, h(\mathtt{D})\, h(\mathtt{L})\, h(\mathtt{E}).$$

Здесь h — хеш-функция, переводящая каждую букву в двухбитовое значение, и мы получаем 12-битовый адрес, записывая рядом шесть пар битов. Затем запрос N**D*E может быть обработан в результате просмотра только 64 списков из 4 096.

Аналогично предположим, что есть 1 000 000 записей, в каждой из которых содержится 10 вторичных ключей, а каждый вторичный ключ имеет большое количество возможных значений. Можно отобразить записи со вторичными ключами $(K_1, K_2, \ldots, K_{10})$ в 20-битовое число

$$h(K_1)\, h(K_2)\, \ldots\, h(K_{10}), \tag{12}$$

где h — хеш-функция, преобразующая каждый вторичный ключ в двухбитовое число, и (12) образуется после размещения этих десяти пар битов в одном 20-битовом числе. Данная схема отображает 1 000 000 записей в $2^{20} = 1\,048\,576$ возможных значений, и отображение, в целом, может рассматриваться как хеш-функция с $M = 2^{20}$. Для разрешения коллизий может использоваться метод цепочек. Чтобы получить все записи с определенными значениями пяти вторичных ключей, потребуется просмотреть только 2^{10} списков, соответствующих пяти неопределенным парам битов в (12). Таким образом, в среднем придется проверить около $1000 = \sqrt{N}$ записей. (Подобный подход был предложен М. Арисавой (M. Arisawa) [*J. Inf. Proc. Soc. Japan* **12** (1971), 163–167] и Б. Двайером (B. Dwyer) [не опубликовано]. Двайер предложил использовать более гибкое по сравнению с (12) отображение, а именно

$$\bigl(h_1(K_1) + h_2(K_2) + \cdots + h_{10}(K_{10})\bigr) \bmod M,$$

где M — любое подходящее число, а h_i — произвольные хеш-функции, возможно, вида $w_i K_i$ для "случайного" w_i.)

Ривест продолжил разработку этой идеи, так что во многих случаях возникает следующая ситуация. Предположим, что имеется $N \approx 2^n$ записей, в каждой из которых содержится m вторичных ключей. Каждая запись отображается в n-битовый хеш-адрес таким образом, что запрос, оставляющий неопределенными значения k ключей, соответствует примерно $N^{k/m}$ хеш-адресам. Все другие методы, обсуждавшиеся ранее в этом разделе (за исключением метода Густафсона), требуют порядка N шагов для получения информации, хотя и с малым коэффициентом пропорциональности; для больших значений N метод Ривеста более быстр и не требует инвертированных файлов.

Но прежде чем применять эту технологию, следует определить подходящее отображение. Ниже рассматривается пример с небольшими параметрами, с $m = 4$ и $n = 3$ и вторичными ключами, принимающими два значения. Можно отобразить 4-битовые записи в восемь адресов следующим образом.

$$
\begin{array}{llll}
*\,0\,0\,1 \to 0 & & *\,1\,1\,0 \to 4 \\
0\,*\,0\,0 \to 1 & & 1\,*\,1\,1 \to 5 \\
1\,0\,*\,0 \to 2 & & 0\,1\,*\,1 \to 6 \\
1\,1\,0\,* \to 3 & & 0\,0\,1\,* \to 7
\end{array}
\tag{13}
$$

Исследование этой таблицы показывает, что все записи, соответствующие запросу $0 * * *$, отображаются в позиции 0, 1, 4, 6 и 7. Точно так же *любой* базовый запрос с тремя символами "*" соответствует в точности пяти позициям, базовый запрос с двумя "*" — трем позициям и с одной "*" — одной или двум позициям, $(8 \times 1 + 24 \times 2)/32 = 1.75$ в среднем. Таким образом, имеем следующее.

Количество неопределенных битов в запросе	Количество позиций поиска
4	$8 = 8^{4/4}$
3	$5 \approx 8^{3/4}$
2	$3 \approx 8^{2/4}$
1	$1.75 \approx 8^{1/4}$
0	$1 = 8^{0/4}$

(14)

Конечно, это всего лишь маленький пример, и в таких случаях гораздо проще осуществлять поиск "в лоб". Этот метод приводит к появлению нетривиальных приложений, поскольку его можно использовать и тогда, когда $m = 4r$ и $n = 3r$, отображая $4r$-битовые записи на $2^{3r} \approx N$ позиций путем разделения вторичных ключей на r групп по 4 бит и применяя (13) к каждой группе. Получающееся отображение имеет требуемое свойство: *запрос, оставляющий k из m бит неопределенными, соответствует примерно $N^{k/m}$ позициям* (см. упр. 6).

В 1997 году А. Э. Брувер (A. E. Brouwer) нашел привлекательный способ сжатия 8 бит до 5 бит при помощи отображения, аналогичного (13). Каждый 8-битовый байт принадлежит в точности одному из следующих 32 классов.

0*000*0*	01*0**11	00*11**1	*11**101
1*000*0*	11*0**11	10*11**1	*11**010
0*010*0*	01*1**11	00*0*01*	*10*0*10
1*010*0*	11*1**11	10*0*01*	*10*1*01
0*10*1*0	0*1*000*	*01*01*1	*0*1001*
1*10*1*0	1*1*000*	*10*10*0	*0*0100*
0*11*1*0	0*0*11*0	*00*011*	*0*011*1
1*11*1*0	1*0*11*0	*11*100*	*0*110*0

(15)

Звездочки в этой конструкции расположены таким образом, что в каждой строке их по 3, а в каждом столбце — по 12. В упр. 18 поясняется, каким образом строятся подобные схемы, сжимающие записи с $m = 4^r$ бит в $n = 3^r$-битовые адреса. На практике используются блоки размером b и $N \approx 2^n b$; случай, когда $b = 1$, использовался выше для упрощения изложения материала.

Ривест предложил также другой простой путь обработки базовых запросов. Предположим, имеется $N \approx 2^{10}$ записей по 30 бит и необходимо ответить на произвольный 30-битовый базовый запрос, подобный (10). В этом случае мы можем просто поделить 30 бит на три 10-битовых поля и поддерживать три отдельные хеш-таблицы размером $M = 2^{10}$. Каждая запись хранится в трех местах, в списках, соответствующих битовым конфигурациям трех полей. При соответствующих усло-

виях в каждом списке будет содержаться около одного элемента. В данном базовом запросе с k неопределенными битами как минимум одно из полей будет иметь $\lfloor k/3 \rfloor$ или меньше неопределенных битов. Следовательно, потребуется просмотреть не более $2^{\lfloor k/3 \rfloor} \approx N^{k/30}$ списков для поиска всех ответов на запрос. Впрочем, можно использовать и любую другую технологию для обработки базовых запросов в выбранных полях.

Обобщенные лучи (tries). Ривест предложил еще один подход, который основан на структурах данных, подобных использованным в разделе 6.3 лучам. Можно предположить, что каждый внутренний узел обобщенного бинарного луча определяет представленные в записи биты. Например, используя приведенные в табл. 1 данные, можно положить корнем луча *Ванилин*. Тогда левый подлуч будет соответствовать тем 16 рецептам, в которых ванилин не нужен, в то время как в правом подлуче собираются 15 рецептов с ванилином. Такое разбиение 16-15 удачно делит файл практически пополам; каждый из подфайлов обрабатывается аналогично, и когда на некоторой стадии подфайл становится достаточно мал, его можно представить в качестве конечного узла.

Для поиска по обобщенному лучу с корнем, определяющим атрибут, которому в запросе соответствует 0 или 1, переходим к поиску в левом или правом подлуче соответственно; если этому атрибуту в запросе соответствует "*", просматриваем оба подлуча.

Предположим, что атрибуты не бинарны, но представлены в бинарной записи. Можно построить луч, рассматривая сначала первый бит атрибута 1, затем — первый бит атрибута 2, ..., первый бит атрибута m, второй бит атрибута 1 и т. д. Такая структура называется m-d-лучом, по аналогии с m-d-деревьями (которые ветвятся в результате сравнения, а не проверки битов). Ф. Флажоле (P. Flajolet) и К. Пуч (C. Puech) показали, что среднее время ответа на частично определенный запрос по случайному m-d-лучу с N узлами составляет $\Theta(N^{k/m})$, если k/m атрибутов не определены [*JACM* **33** (1986), 371–407, §4.1]. Дисперсия этой величины была вычислена В. Шахингером (W. Schachinger), *Random Structures and Algorithms* **7** (1995), 81–95.

Подобный алгоритм может быть распространен на m-мерные версии цифровых деревьев поиска и деревья метода "Патриция" из раздела 6.3. Эти структуры, которые обычно несколько лучше сбалансированы, чем m-d-лучи, были проанализированы в работе P. Kirschenhofer and H. Prodinger, *Random Structures and Algorithms* **5** (1994), 123–134.

***Сбалансированные схемы.** Другой комбинаторный подход к получению информации, основанный на *системах сбалансированных неполных блоков*, был предметом множества исследований. Хотя предмет исследования весьма интересен с математической точки зрения, к сожалению, его преимущества перед другими описанными выше методами до сих пор не доказаны. Здесь будет представлено краткое введение в теорию, чтобы показать изящество результатов, в надежде, что читатели придумают, как внедрить теоретические идеи в практику.

Штейнеровская система троек представляет собой распределение v объектов в неупорядоченные тройки таким образом, что каждая пара объектов встречается ровно в одной тройке. Например, при $v = 7$ имеется ровно одна Штейнеровская система троек.

Тройки	Содержащиеся пары
$\{1,2,4\}$	$\{1,2\}$, $\{1,4\}$, $\{2,4\}$
$\{2,3,5\}$	$\{2,3\}$, $\{2,5\}$, $\{3,5\}$
$\{3,4,6\}$	$\{3,4\}$, $\{3,6\}$, $\{4,6\}$
$\{4,5,0\}$	$\{0,4\}$, $\{0,5\}$, $\{4,5\}$
$\{5,6,1\}$	$\{1,5\}$, $\{1,6\}$, $\{5,6\}$
$\{6,0,2\}$	$\{0,2\}$, $\{0,6\}$, $\{2,6\}$
$\{0,1,3\}$	$\{0,1\}$, $\{0,3\}$, $\{1,3\}$

(16)

Поскольку существует $\frac{1}{2}v(v-1)$ пар объектов и три пары в одной тройке, всего должно быть $\frac{1}{6}v(v-1)$ троек; а так как каждый объект может быть в паре с $v-1$ объектами, каждый объект должен содержаться ровно в $\frac{1}{2}(v-1)$ тройках. Из этих условий следует, что Штейнеровская система троек может существовать тогда и только тогда, когда $\frac{1}{6}v(v-1)$ и $\frac{1}{2}(v-1)$ представляют собой целые числа. Это эквивалентно тому, что v должно быть нечетно и не быть равным 2 по модулю 3; таким образом,

$$v \bmod 6 = 1 \text{ или } 3. \tag{17}$$

И обратно, в 1847 году Т. П. Киркман (T. P. Kirkman) доказал, что Штейнеровская система троек существует для всех $v \geq 1$, удовлетворяющих (17). Его интересное построение приведено в упр. 10.

Штейнеровские системы троек могут использоваться для снижения избыточности индексов комбинированных инвертированных файлов. Вернемся, например, к рассматривавшимся ранее рецептам печенья из табл. 1 и конвертируем крайний справа столбец в тридцать первый атрибут, который равен 1, если требуется какой-либо специальный ингредиент, и равен 0 в противном случае. Предположим, что необходимо ответить на все включающие запросы о парах атрибутов наподобие "В каких рецептах используются и кокосовый орех, и изюм?". Можно было бы построить инвертированные файлы для всех $\binom{31}{2} = 465$ возможных запросов. Однако они займут очень много места, так как, например, "Печенье с перцем" будет содержаться в $\binom{17}{2} = 136$ списках, а запись, включающая 31 атрибут, будет содержаться в каждом списке! Штейнеровская система троек позволяет немного улучшить ситуацию. В случае 31 объекта существует Штейнеровская система из 155 троек, в которой каждая пара объектов встречается ровно в одной тройке. Каждой тройке $\{a,b,c\}$ можно назначить четыре списка: первый — для всех записей с атрибутами $\{a,b,\bar{c}\}$ (т. е. a, b, но не c), второй — для $\{a,\bar{b},c\}$, третий — для $\{\bar{a},b,c\}$ и четвертый — для записей со всеми тремя атрибутами $\{a,b,c\}$. Таким образом гарантируется, что никакая запись не будет включена более чем в 155 инвертированных списков, и сохраняется пространство, когда запись имеет три атрибута, соответствующих тройке системы.

Системы троек представляют собой частный случай систем блоков из трех или более объектов. Например, тот же 31 объект может быть распределен по шестеркам так, чтобы каждая пара объектов оказывалась ровно в одной шестерке:

$$\{0,4,16,21,22,24\},\ \{1,5,17,22,23,25\},\ \ldots,\ \{30,3,15,20,21,23\}. \tag{18}$$

(Эта конструкция создана из первого блока путем сложения по модулю 31. Чтобы проверить, что она обладает указанными свойствами, обратим внимание на сле-

дующий факт: 30 значений $(a_i - a_j) \bmod 31, i \neq j$ различны при $(a_1, a_2, \ldots, a_6) = (0, 4, 16, 21, 22, 24)$. Для поиска шестерки, содержащей пару (x, y), выберем i и j такими, что $a_i - a_j \equiv x - y$ (по модулю 31). Теперь, если $k = (x - a_i) \bmod 31$, имеем $(a_i + k) \bmod 31 = x$ и $(a_j + k) \bmod 31 = y$.)

Можно использовать конструкцию, приведенную выше, для хранения инвертированных списков таким образом, чтобы ни одна запись не появилась более 31 раза. Каждая шестерка $\{a, b, c, d, e, f\}$ ассоциирована с 57 списками всех возможных для записей, имеющих два или более атрибутов a, b, c, d, e, f, а именно — $\{a, b, \bar{c}, \bar{d}, \bar{e}, \bar{f}\}$, $\{a, \bar{b}, c, \bar{d}, \bar{e}, \bar{f}\}$, $\ldots$, $\{a, b, c, d, e, f\}$. Ответом на любой включающий запрос по двум атрибутам является объединение без пересечений 16 подходящих списков соответствующей шестерки. Так, "Печенье с перцем" войдет в 29 блоков из 31, поскольку эта запись имеет по два из шести атрибутов во всех шестерках, кроме $\{19, 23, 4, 9, 10, 12\}$ и $\{13, 17, 29, 3, 4, 6\}$, если перенумеровать столбцы от 0 до 30.

Теория построения блоков и связанные с ней вопросы детально рассмотрены в книге Маршалла Холла (мл.) (Marshall Hall, Jr.) *Combinatorial Theory* (Waltham, Mass.: Blaisdell, 1967). Хотя такие комбинаторные построения очень красивы, их основное приложение к задачам получения информации сводится к уменьшению избыточности при построении инвертированных списков. Однако в работе David K. Chow, *Information and Control* **15** (1969), 377–396, замечено, что такое уменьшение может быть достигнуто и без использования комбинаторных конструкций.

Краткая история и библиография. Первой опубликованной статьей, посвященной вопросам технологии получения информации по вторичным ключам, явилась статья L. R. Johnson, *CACM* **4** (1961), 218–222. Многосписочная система была независимо разработана в то же время Ноа Ш. Прайзом (Noah S. Prywes), Г. Дж. Греем (H. J. Gray), В. И. Ландауэром (W. I. Landauer), Д. Лефковицем (D. Lefkowitz) и С. Литвином (S. Litwin) (см. *IEEE Trans. on Communication and Electronics* **82** (1963), 488–492). Еще одна ранняя публикация, оказавшая влияние на более поздние работы, принадлежит Д. Р. Дэвису (D. R. Davis) и Э. Д. Лину (A. D. Lin), *CACM* **8** (1965), 243–246.

С тех пор количество публикаций на данную тему быстро увеличивается, однако почти все они посвящены таким не имеющим отношения к нашей книге вопросам, как пользовательский интерфейс и использование тех или иных языков программирования. Кроме уже указанных статей, автор считает, что наиболее полезными при написании этого раздела (начиная с 1972 года — времени создания настоящего раздела) были следующие работы: Jack Minker and Jerome Sable, *Ann. Rev. of Information Science and Technology* **2** (1967), 123–160; Robert E. Bleier, *Proc. ACM Nat. Conf.* **22** (1967), 41–49; Jerome A. Feldman and Paul D. Rovner, *CACM* **12** (1969), 439–449; Burton H. Bloom, *Proc. ACM Nat. Conf.* **24** (1969), 83–95; H. S. Heaps and L. H. Thiel, *Information Storage and Retrieval* **6** (1970), 137–153; Vincent Y. Lum and Huei Ling, *Proc. ACM Nat. Conf.* **26** (1971), 349–356. Хороший обзор ручных картотек содержится в работе C. P. Bourne, *Methods of Information Handling* (New York: Wiley, 1963), Chapter 5. Сбалансированные схемы были первоначально разработаны в 1965 году Ч. Т. Абрахамом (C. T. Abraham), С. П. Гошем (S. P. Ghosh) и Д. К. Рэй-Чаудури (D. K. Ray-Chaudhuri); см. статью R. C. Bose and Gary G. Koch, *SIAM J. Appl. Math.* **17** (1969), 1203–1214.

Большинство классических алгоритмов для многоатрибутных данных, практическая важность которых общеизвестна, обсуждались в этом разделе; однако в следующее издание настоящей книги планируется добавить еще несколько тем, включая следующие.

- Э. М. Мак-Крейт (E. M. McCreight) ввел понятие *приоритетные деревья поиска* (*priority search trees*) [*SICOMP* **14** (1985), 257–276], которые разработаны специально для представления пересечений динамически изменяющихся семейств интервалов и обработки запросов диапазонов в форме "Найти все записи с $x_0 \le x \le x_1$ и $y \le y_1$" (заметьте, что нижняя грань y должна быть равной $-\infty$, но x может быть ограничен с обеих сторон).
- М. Л. Фредман (M. L. Fredman) нашел ряд фундаментальных нижних граней, показывающих, что последовательность N смешанных вставок, удалений и k-мерных запросов диапазонов требуют в худшем случае выполнения $\Omega\big(N(\log N)^k\big)$ операций, независимо от используемых структур данных. См. *JACM* **28** (1981), 696–705; *SICOMP* **10** (1981), 1–10; *J. Algorithms* **1** (1981), 77–87.

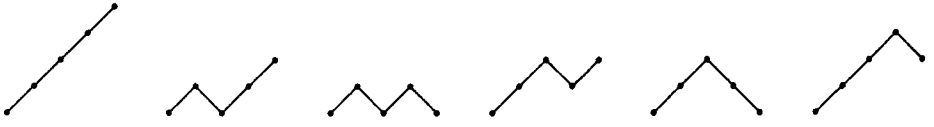
Базовые алгоритмы поиска соответствий шаблону (pattern matching) и приближенных соответствий шаблону в текстовых строках будут обсуждаться в главе 9.

Интересно отметить, что мозг человека гораздо лучше компьютера справляется с поиском информации по вторичным ключам. Люди достаточно легко распознают лица, мелодии и т. п. по фрагментам информации, в то время как для компьютера это практически непосильная задача. Таким образом, вполне вероятно, что в один прекрасный день будет найден совершенно новый подход к решению задач, связанных с поиском информации по вторичным ключам, который сделает это примечание, да и весь текущий раздел, безнадежно устаревшим.

УПРАЖНЕНИЯ

▶ **1.** [*M27*] Пусть $0 \le k \le n/2$. Докажите, что следующая конструкция дает $\binom{n}{k}$ перестановок множества $\{1, 2, \ldots, n\}$, таких, что каждое t-элементное подмножество множества $\{1, 2, \ldots, n\}$ встречается в качестве первых t элементов как минимум одной перестановки при $t \le k$ или $t \ge n-k$. Рассмотрим путь на плоскости из точки $(0, 0)$ в точку (n, r), где $r \ge n - 2k$, в котором i-й шаг выполняется из точки $(i-1, j)$ в точку $(i, j+1)$ или $(i, j-1)$; последняя возможность разрешена только при $j \ge 1$, так что путь никогда не проходит ниже оси x. Существует ровно $\binom{n}{k}$ таких путей. Для каждого из них соответствующая перестановка строится с использованием трех первоначально пустых списков следующим образом: для $i = 1, 2, \ldots, n$, если i-й шаг идет вверх, поместим число i в список B; если шаг идет вниз, поместим i в список A и переместим максимальный на текущий момент список B в список C. Результирующая перестановка представляет собой содержимое списка A, затем списков B и C, причем содержимое каждого списка приводится в порядке возрастания.

Например, при $n = 4$ и $k = 2$ такая процедура определяет шесть путей и перестановок.



|1 2 3 4| 2|3 4|1 2 4||1 3 3|1 4|2 3 4||1 2 4|1 2|3

(Вертикальные линии отделяют списки A, B и C. Эти шесть перестановок соответствуют составным атрибутам в (8).)

Указание. Представьте каждое t-элементное подмножество S в виде пути, который идет из $(0,0)$ в $(n, n-2t)$, i-ый шаг которого идет из $(i-1, j)$ в $(i, j+1)$ при $i \notin S$ и в $(i, j-1)$ при $i \in S$. Преобразуйте каждый такой путь в путь описанного выше вида.

2. [*M25*] (Сакти П. Гош (Sakti P. Ghosh).) Найдите минимально возможную длину l списка $r_1 r_2 \ldots r_l$ ссылок на записи, такого, что множество всех ответов на любой из включающих запросов ****1**, ***1***, **1****, ***11**, **1*1**, **11***, **111** по трем вторичным ключам с двумя возможными значениями оказывается расположенным в последовательных позициях $r_i \ldots r_j$.

3. [*19*] Какие включающие запросы к табл. 2 вызовут ложное выпадение (a) старинного сахарного печенья, (b) овсяных палочек с финиками?

4. [*M30*] Найдите точные формулы для вероятностей в (11), предполагая, что каждая запись имеет r различных атрибутов, случайным образом выбираемых из $\binom{n}{k}$ k-битовых кодов в n-битовом поле, и что запрос включает q различных (но в остальном случайных) атрибутов (не волнуйтесь, если формулы не будут упрощаться).

5. [*40*] Проэкспериментируйте с различными путями снижения избыточности текста при использовании метода Харрисона для поиска подстрок.

▶ **6.** [*M20*] Общее количество m-битовых базовых запросов с t определенными битами составляет $s = \binom{m}{t}2^t$. Если комбинаторная хеш-функция наподобие (13) конвертирует такие запросы в позиции $l_1, l_2, \ldots, l_s$ соответственно, то среднее количество позиций на запрос составляет $L(t) = (l_1 + l_2 + \cdots + l_s)/s$ (например, в (13) получим $L(3) = 1.75$).

Рассмотрим теперь составную хеш-функцию над $(m_1 + m_2)$-битовым полем, построенную путем отображения первых m_1 бит с помощью одной хеш-функции и оставшихся m_2 с помощью другой, а $L_1(t)$ и $L_2(t)$ — соответствующие средние количества позиций на запрос. Найдите формулу, выражающую значение $L(t)$ в случае составной функции через L_1 и L_2.

7. [*M24*] (Р. Л. Ривест (R. L. Rivest).) Найдите функции $L(t)$, определенные в предыдущем упражнении, для следующих комбинаторных хеш-функций.

(a) $m = 3$, $n = 2$	(b) $m = 4$, $n = 2$
$0\,0\,* \to 0$	$0\,0\,*\,* \to 0$
$1\,*\,0 \to 1$	$*\,1\,*\,0 \to 1$
$*\,1\,1 \to 2$	$*\,1\,1\,1 \to 2$
$1\,0\,1 \to 3$	$1\,0\,1\,* \to 2$
$0\,1\,0 \to 3$	$*\,1\,0\,1 \to 3$
	$1\,0\,0\,* \to 3$

8. [*M32*] (Р. Л. Ривест.) Рассмотрим множество $Q_{t,m}$ всех $2^t\binom{m}{t}$ базовых m-битовых запросов типа (10), в которых определено ровно t бит. Дано множество m-битовых записей S. Пусть $f_t(S)$ описывает количество запросов в $Q_{t,m}$, в ответах на которые содержатся члены множества S, а $f_t(s,m)$ представляет собой минимум $f_t(S)$ по всем таким множествам S с s элементами при $0 \le s \le 2^m$. По определению $f_t(0,0) = 0$ и $f_t(1,0) = \delta_{t0}$.

a) Докажите, что для всех $t \ge 1$ и $m \ge 1$ при $0 \le s \le 2^m$

$$f_t(s,m) = f_t(\lceil s/2 \rceil, m-1) + f_{t-1}(\lceil s/2 \rceil, m-1) + f_{t-1}(\lfloor s/2 \rfloor, m-1).$$

b) Рассмотрим некоторую комбинаторную хеш-функцию h, отображающую 2^m возможных записей на 2^n списков, причем каждый список соответствует 2^{m-n} записям. Если все запросы $Q_{t,m}$ равновероятны, то среднее количество проверяемых списков на запрос составляет $1/2^t\binom{m}{t}$, умноженное на

$$\sum_{Q\in Q_{t,m}} (\text{списки, проверяемые для } Q) =$$

$$= \sum_{\text{списки } S} (\text{запросы } Q_{t,m}, \text{ относящиеся к } S) \geq 2^n f_t(2^{m-n}, m).$$

Покажите, что h оптимальна в том смысле, что нижняя грань достигается, когда каждый из списков представляет собой "субкуб"; другими словами, покажите, что в случае, когда каждый список соответствует множеству записей, удовлетворяющих базовому запросу с ровно n определенными битами, неравенство превращается в равенство.

9. [*M20*] Докажите, что если $v = 3^n$, то множество всех троек вида

$$\{(a_1 \ldots a_{k-1}\, 0\, b_1 \ldots b_{n-k})_3,\ (a_1 \ldots a_{k-1}\, 1\, c_1 \ldots c_{n-k})_3, (a_1 \ldots a_{k-1}\, 2\, d_1 \ldots d_{n-k})_3\},$$

$1 \leq k \leq n$, где a, b, c и d принимает значения 0, 1 или 2 и $b_j + c_j + d_j \equiv 0$ (по модулю 3) при $1 \leq j \leq n-k$ образует Штейнеровскую систему троек.

10. [*M32*] (Томас П. Киркман (Thomas P. Kirkman), *Cambridge and Dublin Math. Journal* **2** (1847), 191–204.) Назовем *системой троек Киркмана* порядка v такое упорядочение $v+1$ объектов $\{x_0, x_1, \ldots, x_v\}$ в тройки, при котором каждая пара $\{x_i, x_j\}, i \neq j$ встречается ровно в одной тройке, за исключением v пар $\{x_i, x_{(i+1) \bmod v}\}, 0 \leq i < v$, которые не встречаются в тройках. Например,

$$\{x_0, x_2, x_4\},\ \{x_1, x_3, x_4\}$$

представляет собой систему троек Киркмана порядка 4.

a) Докажите, что система троек Киркмана может существовать только для $v \bmod 6 = 0$ или 4.

b) Дана Штейнеровская система троек S над v объектами $\{x_1, \ldots, x_v\}$. Докажите, что следующее построение дает другую Штейнеровскую систему троек S' над $2v+1$ объектами и систему троек Киркмана K' порядка $2v-2$. Тройки S' представляют собой все тройки из S плюс

   i) $\{x_i, y_j, y_k\}$, где $j + k \equiv i$ (по модулю v) и $j < k$, $1 \leq i, j, k \leq v$;

   ii) $\{x_i, y_j, z\}$, где $2j \equiv i$ (по модулю v), $1 \leq i, j \leq v$.

   Тройки системы K' представляют собой множество троек S' минус все тройки, содержащие y_1 и/или y_v.

c) Дана система троек Киркмана K на множестве $\{x_0, x_1, \ldots, x_v\}$, где $v = 2u$. Докажите, что следующее построение дает Штейнеровскую систему троек S' над $2v+1$ объектами и систему троек Киркмана K' порядка $2v-2$. Тройки S' представляют собой тройки K плюс

   i) $\{x_i, x_{(i+1) \bmod v}, y_{i+1}\}$, $0 \leq i < v$;

   ii) $\{x_i, y_j, y_k\}$, $j + k \equiv 2i + 1$ (по модулю $v-1$), $1 \leq j < k-1 \leq v-2$, $1 \leq i \leq v-2$;

   iii) $\{x_i, y_j, y_v\}$, $2j \equiv 2i + 1$ (по модулю $v-1$), $1 \leq j \leq v-1$, $1 \leq i \leq v-2$;

   iv) $\{x_0, y_{2j}, y_{2j+1}\}$, $\{x_{v-1}, y_{2j-1}, y_{2j}\}$, $\{x_v, y_j, y_{v-j}\}$, $1 \leq j < u$;

   v) $\{x_v, y_u, y_v\}$.

   Тройки K' представляют собой множество троек S' минус все тройки, содержащие y_1 и/или y_{v-1}.

d) Используйте предыдущие результаты для доказательства того, что система троек Киркмана порядка v существует для всех $v \geq 0$ вида $6k$ или $6k+4$ и Штейнеровская система троек над v объектами существует для всех $v \geq 1$ вида $6k+1$ или $6k+3$.

11. [*M25*] В тексте раздела описано использование Штейнеровских систем троек в связи с включающими запросами. Для расширения их применения на все базовые запросы естественно определить следующую концепцию. *Комплементарной системой троек* порядка v является такое размещение $2v$ объектов $\{x_1, \ldots, x_v, \bar{x}_1, \ldots, \bar{x}_v\}$ в тройках, при котором каждая пара объектов встречается ровно в одной тройке, за исключением комплементарных пар $\{x_i, \bar{x}_i\}$, никогда не встречающихся вместе. Например,

$$\{x_1, x_2, x_3\},\ \{x_1, \bar{x}_2, \bar{x}_3\},\ \{\bar{x}_1, x_2, \bar{x}_3\},\ \{\bar{x}_1, \bar{x}_2, x_3\}$$

представляет собой комплементарную систему троек третьего порядка.

Докажите, что комплементарные системы троек порядка v существуют для всех $v \geq 0$, не имеющих вид $3k + 2$.

12. [*M23*] Продолжая упр. 11, постройте комплементарную систему *четверок* порядка 7.

13. [*M25*] Постройте систему четверок с $v = 4^n$ элементами, аналогичную системе троек из упр. 9.

14. [*28*] Обсудите проблему удаления узлов из четревьев, k-d-деревьев и почтовых деревьев, подобных показанному на рис. 45.

15. [*HM30*] (П. Элиас (P. Elias).) Дана большая коллекция m-битовых записей. Предположим, что необходимо найти запись, ближайшую к данному аргументу поиска в том смысле, что согласовано наибольшее количество битов. Разработайте алгоритм эффективного решения этой задачи в предположении, что задан m-битовый код из 2^n элементов, исправляющий t ошибок, и что каждая запись хешируется в один из 2^n списков, соответствующих ближайшему кодовому слову.

▶ **16.** [*25*] (В. Х. Кац (W. H. Kautz) и Р. К. Синглтон (R. C. Singleton).) Покажите, что Штейнеровская система троек порядка v может использоваться для построения $v(v-1)/6$ v-битовых кодовых слов, таких, что никакое слово не содержится в суперпозиции двух других.

▶ **17.** [*M30*] Рассмотрите следующий путь сведения $(2n+1)$-битовых ключей $a_{-n} \ldots a_0 \ldots a_n$ к $(n+1)$-битовому блоку адресов $b_0 \ldots b_n$:

$$b_0 \leftarrow a_0;$$
$$\text{если } b_{k-1} = 0, \text{ то } b_k \leftarrow a_{-k}, \text{ иначе } b_k \leftarrow a_k \text{ для } 1 \leq k \leq n.$$

a) Опишите ключи, появляющиеся в блоке $b_0 \ldots b_n$.

b) Чему равно наибольшее количество блоков, которые необходимо проверить при базовом запросе с t определенными битами?

▶ **18.** [*M35*] (*Конструирование ассоциативного блока.*) Множество m-элементных наборов типа (13) с ровно $m - n$ звездочками в каждой из 2^n строк называется ABD(m, n)*, если в каждом столбце содержится одно и то же количество звездочек и если каждая пара строк имеет "несоответствие" (0 против 1) в некотором столбце. Каждое m-битовое бинарное число будет в таком случае соответствовать в точности одной строке. Например, (13) является ABD$(4, 3)$.

a) Докажите, что ABD(m, n) невозможно, кроме случаев, когда m является делителем $2^{n-1}n$ и $n^2 \geq 2m(1 - 2^{-n})$.

b) Будем говорить, что строка ABD имеет *нечетный паритет*, если в ней содержится нечетное количество единиц. Покажите, что для любого выбора $m - n$ столбцов в ABD(m, n) количество строк с нечетным паритетом с "*" в этих столбцах равно количеству строк с четным паритетом. В частности, каждое возможное расположение символов "*" должно встречаться в четном количестве строк.

---

* От *Associative Block Designs*. — *Прим. перев.*

c) Найдите ABD(4, 3), которое не может быть получено из (13) путем перестановки и/или дополнения столбцов.
d) Постройте ABD(16, 9).
e) Постройте ABD(16, 10). Начните с ABD(16, 9) из п. (d) вместо ABD(8, 5) из (15).

19. [*M22*] Проанализируйте ABD(8, 5) из (15) так же, как (13) было проанализировано в (14). Сколько из 32 позиций должно быть проверено при запросе с k неопределенными битами в среднем и в наихудшем случаях?

20. [*M47*] Найдите все ABD(m, n) при $n = 5$ или $n = 6$.

Новый раздел 6.6 в следующем издании книги планируется посвятить постоянным структурам данных. Эти структуры позволяют представить измененную информацию таким образом, что история изменений может быть эффективно восстановлена. Иными словами, можно сделать множество вставок и удалений, но, тем не менее, оставаться способными выполнить поиск так, как будто обновлений после некоторого данного момента времени вовсе не было. Пока что в настоящее издание включены лишь ссылки на литературу по этой теме:

- J. K. Mullin, *Comp. J.* **24** (1981), 367–373;
- M. H. Overmars, *Lecture Notes in Comp. Sci.* **156** (1983), Chapter 9;
- E. W. Myers, *ACM Symp. Principles of Prog. Lang.* **11** (1984), 66–75;
- B. Chazelle, *Information and Control* **63** (1985), 77–99;
- D. Dobkin and J. I. Munro, *J. Algorithms* **6** (1985), 455–465;
- R. Cole, *J. Algorithms* **7** (1986), 202–220;
- D. Field, *Information Processing Letters* **24** (1987), 95–96;
- C. W. Fraser and E. W. Myers, *ACM Trans. Prog. Lang. and Systems* **9** (1987), 277–295;
- J. R. Driscoll, N. Sarnak, D. D. Sleator, and R. E. Tarjan, *J. Comp. Syst. Sci.* **38** (1989), 86–124;
- R. B. Dannenberg, *Software Practice & Experience* **20** (1990), 109–132;
- J. R. Driscoll, D. D. K. Sleator, and R. E. Tarjan, *JACM* **41** (1994), 943–959.

Таблицы инструкций (программы) будут создаваться математиками с опытом вычислительной работы и, вероятно, с определенными способностями к решению головоломок. Перевод на некоторой стадии каждого известного процесса в форму таблиц инструкций, скорее всего, будет значительной частью такой работы. ... Процесс построения таблиц инструкций должен очаровывать. Реальной опасности стать слугой машины нет, ибо любой более или менее механический процесс можно передать самой машине.

— АЛАН М. ТЬЮРИНГ (ALAN M. TURING) (1945)

ПРИЛОЖЕНИЕ А

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−
$1^\circ = \pi/180 =$	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−
$\sqrt{\pi} = \Gamma(1/2) =$	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−

Таблица 2

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−*

Несколько интересных констант без общего названия возникли в связи с анализом алгоритмов сортировки и выбора. Эти константы вычислены с 40 десятичными знаками в 5.2.3–(19) и 6.5–(6) и в ответах к упр. 5.2.3–27, 5.2.4–13, 5.2.4–23, 6.2.2–49, 6.2.3–7, 6.2.3–8, 6.3–26 и 6.3–27.

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),

$$H_{p/q} = \frac{q}{p} - \frac{\pi}{2}\cot\frac{p}{q}\pi - \ln 2q + 2\sum_{1\leq n<q/2}\cos\frac{2pn}{q}\pi\cdot\ln\sin\frac{n}{q}\pi.$$

ПРИЛОЖЕНИЕ Б

ОСНОВНЫЕ ОБОЗНАЧЕНИЯ

Буквы в формулах, если дополнительно не оговорено, имеют следующий смысл.

j, k	Арифметическое выражение, значением которого является целое число
m, n	Арифметическое выражение, значением которого является неотрицательное целое число
x, y	Арифметическое выражение, принимающее действительное значение
z	Арифметическое выражение, принимающее комплексное значение
f	Функция, принимающая действительное или комплексное значение
P	Выражение, значение которого — указатель (либо Λ, либо адрес компьютера)
S, T	Множество или мультимножество
α, β	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 которой 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 компьютера, 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

Обозначение	Значение	Раздел
$X \Leftarrow \mathsf{S}$	Взято из S в X: присвоить $X \leftarrow \mathrm{top}(\mathsf{S})$ и затем удалить $\mathrm{top}(\mathsf{S})$ из непустого стека S	2.2.1
$\mathsf{S} \Leftarrow X$	Поместить X в S: вставить значение X в качестве нового входного значения в вершину стека S	2.2.1
$(B \Rightarrow E;\ E')$	Условное выражение: означает E, если B истинно, и E', если B ложно	
$[B]$	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
$\sum\limits_{R(k)} f(k)$	Сумма всех $f(k)$, таких, что значение k — целое и выполняется соотношение $R(k)$	1.2.3
$\prod\limits_{R(k)} f(k)$	Произведение всех $f(k)$, таких, что значение k — целое и выполняется соотношение $R(k)$	1.2.3
$\min\limits_{R(k)} f(k)$	Минимальное значение из всех $f(k)$, таких, что значение k — целое и выполняется соотношение $R(k)$	1.2.3
$\max\limits_{R(k)} f(k)$	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: $\left(j = k = 0 \Rightarrow 0;\ \max\limits_{d \backslash j,\, d \backslash k} d\right)$	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 к α элемент	
x^y	x в степени y (когда x — положительное число)	1.2.2
x^k	x в степени k: $\left(k \geq 0 \Rightarrow \prod\limits_{0 \leq j < k} x;\quad 1/x^{-k}\right)$	1.2.2

Обозначение	Значение	Раздел
$x^{\overline{k}}$	$\Gamma(x+k)/\Gamma(x) = \left(k \geq 0 \Rightarrow \prod_{0 \leq j < k} (x+j);\quad 1/(x+k)^{\overline{-k}}\right)$	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
$\binom{x}{k}$	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: $\sum_{1 \leq k_1 \leq k_2 \leq \cdots \leq k_{n-m} \leq m} k_1 k_2 \ldots k_{n-m}$	1.2.6
$\{a \mid R(a)\}$	Множество всех a, таких, что выполняется соотношение $R(a)$	
$\{a_1, \ldots, a_n\}$	Множество или мультимножество $\{a_k \mid 1 \leq k \leq n\}$	
$\{x\}$	Дробная часть (используется, когда x — действительное число, а не множество): $x - \lfloor x \rfloor$	1.2.11.2
$[a\,..\,b]$	Замкнутый интервал: $\{x \mid a \leq x \leq b\}$	1.2.2
$(a\,..\,b)$	Открытый интервал: $\{x \mid a < x < b\}$	1.2.2
$[a\,..\,b)$	Полузамкнутый интервал: $\{x \mid a \leq x < b\}$	1.2.2
$(a\,..\,b]$	Полуоткрытый интервал: $\{x \mid a < x \leq b\}$	1.2.2
$\lvert S \rvert$	Число элементов множества S	
$\lvert x \rvert$	Абсолютная величина x: $(x \geq 0 \Rightarrow x;\ -x)$	
$\lvert \alpha \rvert$	Длина α	
$\lfloor x \rfloor$	Наибольшее целое число, не превосходящее x: $\max_{k \leq x} k$	1.2.4
$\lceil x \rceil$	Наименьшее целое число $> x$: $\min_{k \geq x} k$	1.2.4
$x \bmod y$	x по модулю y: $(y = 0 \Rightarrow x;\ x - y\lfloor x/y \rfloor)$	1.2.4

Обозначение	Значение	Раздел
$x \equiv x'$ (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
$\Theta(f(n))$	Тэта 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 $b \neq 1$): 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
$\langle X_n \rangle$	Бесконечная последовательность X_0, X_1, X_2, ... (здесь буква 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: $\sum\limits_{1 \le k \le n} 1/k^x$	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: $\lim_{n \to \infty} H_n^{(x)}$ (когда $x > 1$)	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)	

Обозначение	Значение	Раздел
$\emptyset$	Пустое множество (множество, не содержащее элементов)	
ϕ	Золотое сечение: $\frac{1}{2}(1+\sqrt{5})$	1.2.8
$\varphi(n)$	Функция Эйлера: $\sum\limits_{0\le k<n}[k\perp n]$	1.2.4
$x\approx y$	x приближенно равно y	1.2.5
$\Pr(S(X))$	Вероятность того, что утверждение $S(X)$ справедливо для случайных величин X	1.2.10
$\mathrm{E}\,X$	Математическое ожидание (среднее значение) случайной величины X: $\sum_x x\Pr(X=x)$, если X — дискретная случайная величина	1.2.10
$\mathrm{mean}(g)$	Среднее значение распределения вероятностей, которое задано производящей функцией g: $g'(1)$	1.2.10
$\mathrm{var}(g)$	Дисперсия распределения вероятностей, которое задано производящей функцией g: $g''(1)+g'(1\rangle-g'(1)^2$	1.2.10
$(\min x_1,\ \mathrm{ave}\ x_2,$ $\max x_3,\ \mathrm{dev}\ x_4)$	Случайная величина с минимальным значением x_1, средним значением (математическим ожиданием) x_2, максимальным значением x_3, среднеквадратичным отклонением x_4	1.2.10
$\Re z$	Действительная часть z	1.2.2
$\Im z$	Мнимая часть z	1.2.2
$\overline{z}$	Комплексное число, сопряженное к z: $\Re z-i\,\Im z$	1.2.2
$(\ldots a_1a_0.a_{-1}\ldots)_b$	Представление числа в позиционной системе счисления с основанием b: $\sum_k a_kb^k$	4.1
$//x_1,x_2,\ldots,x_n//$	Цепная дробь: $1/(x_1+1/(x_2+1/(\cdots+1/(x_n)\ldots)))$	4.5.3
$\alpha\top\beta$	Соединительное произведение	5.1.2
$S\uplus T$	Сумма мультимножеств, т. е. $\{a,b\}\uplus\{a,c\}=\{a,a,b,c\}$	4.6.3
$f(x)\big\|_a^b$	Приращение функции: $f(b)-f(a)$	
∎	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

Обозначение	Значение	Раздел
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, $0 \leq L \leq R \leq 5$	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

ОТВЕТЫ К УПРАЖНЕНИЯМ

"Нет, довольно! — сказал возмущенный отец.
— Есть граница любому терпенью.
Если пятый вопрос ты задашь наконец,
*Сосчитаешь ступень за ступенью!"**

— ЛЬЮИС КЭРРОЛ (LEWIS CARROLL), *Алиса в стране чудес*, гл. 5. (1862)

ПРИМЕЧАНИЯ К УПРАЖНЕНИЯМ

1. Задача средней сложности для читателей с математическим уклоном.

3. 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 5

1. Пусть $p(1)\ldots p(n)$ и $q(1)\ldots q(n)$ — две различные перестановки, удовлетворяющие этим условиям, и пусть i — минимальный индекс, при котором $p(i) \neq q(i)$. Тогда $p(i) = q(j)$ при некоторых $j > i$, а $q(i) = p(k)$ при некоторых $k > i$. Поскольку $K_{p(i)} \leq K_{p(k)} = K_{q(i)} \leq K_{q(j)} = K_{p(i)}$, имеем $K_{p(i)} = K_{q(i)}$; а так как сортировка устойчива, то $p(i) < p(k) = q(i) < q(j) = p(i)$, чего быть не может.

2. Да, если все операции сортировки были *устойчивыми*. (В противном случае этого утверждать нельзя.) Ясно, что Алиса и Крис получили одинаковые результаты; то же самое получил и Билл, так как вследствие устойчивости при равных больших ключах малые ключи расположились в порядке неубывания.

Формальное доказательство: пусть после сортировки по малым ключам Билл получил $R_{p(1)}\ldots R_{p(N)} = R'_1\ldots R'_N$, а после второй сортировки по большим ключам — $R'_{q(1)}\ldots R'_{q(N)} = R_{p(q(1))}\ldots R_{p(q(N))}$. Нужно показать, что при $1 \leq i < N$

$$\left(K_{p(q(i))}, k_{p(q(i))}\right) \leq \left(K_{p(q(i+1))}, k_{p(q(i+1))}\right).$$

Если $K_{p(q(i))} \neq K_{p(q(i+1))}$, то $K_{p(q(i))} < K_{p(q(i+1))}$; если же $K_{p(q(i))} = K_{p(q(i+1))}$, то $K'_{q(i)} = K'_{q(i+1)}$, и тогда $q(i) < q(i+1)$. Следовательно, $k'_{q(i)} \leq k'_{q(i+1)}$, т. е. $k_{p(q(i))} \leq k_{p(q(i+1))}$.

3. Всегда можно собрать вместе записи с одинаковыми ключами, сохранив при этом их относительное расположение; тогда в последующих операциях каждая такая группа рассматривается как неделимое целое. Следовательно, можно считать, что все ключи различны. Пусть $a < b < c < a$; эти три ключа можно упорядочить так: abc, bca или cab. Далее, если упорядочены $N-1$ различных ключей тремя способами, можно это сделать и для N ключей, поскольку при $K_1 < \cdots < K_{N-1} > K_N$ либо $K_{i-1} < K_N < K_i$ для некоторого i, либо $K_N < K_1$.

* *Перевод С. Я. Маршака.*

4. Сначала сравним слова, не обращая внимания на регистр символов, затем воспользуемся регистром символов для уточнения порядка. Точнее, заменим каждое слово α парой (α', α), в которой α' формируется из α подстановкой A $\to$ a, ..., Z $\to$ z; затем рассортируем сформированные пары лексикографически. В результате получим, например, tex < Tex < TeX < TEX < text.

В словарях также встречаются слова с диатоническими знаками, префиксами, суффиксами и аббревиатуры; например,

$$\text{a} < \text{A} < \text{Å} < \text{a-} < \text{a.} < \text{-a} < \text{A-} < \text{A.} < \text{aa} < \text{a.a.}$$
$$< \text{ãa} < \text{ãã} < \text{AA} < \text{A.A.} < \text{AAA} < \cdots < \text{zz} < \text{Zz.} < \text{ZZ} < \text{zzz} < \text{ZZZ.}$$

В таком более общем случае получим α' отображением ã $\to$ a, Å $\to$ a и т. д., отбрасывая знаки дефиса и точки.

5. Положим $\rho(0) = 0$ и $\rho((1\alpha)_2) = 1\rho(|\alpha|)\alpha$; здесь $(1\alpha)_2$ является обычным двоичным представлением положительного целого числа, а $|\alpha|$ — длиной строки α. Тогда $\rho(1) = 10$, $\rho(2) = 1100$, $\rho(3) = 1101$, $\rho(4) = 1110000$, ..., $\rho(1009) = 111101001111110001$, ..., $\rho(65536) = 1^5 0^{24}$, ..., $\rho(2^{65536}) = 1^6 0^{65560}$ и т. д. Длина $\rho(n)$ равна

$$|\rho(n)| = \lambda(n) + \lambda(\lambda(n)) + \lambda(\lambda(\lambda(n))) + \cdots + \lg^* n + 1,$$

где $\lambda(0) = 0$, $\lambda(n) = \lfloor \lg n \rfloor$ для $n \geq 1$, а $\lg^* n$ есть наименьшее целое $m \geq 0$, такое, что $\lambda^{[m]}(n) = 0$. [Эта конструкция предложена В. И. Левенштейном, *Проблемы кибернетики* **20** (1968), 173–179; см. также Д. Э. Кнут в *The Mathematical Gardner*, edited by D. A. Klarner (Belmont, California: Wadsworth International, 1981), 310–325.]

6. Возможно переполнение, которое может исказить результаты сравнения. Следовало написать, например, "LDA A; CMPA B" и проверить индикатор сравнения. (Невозможность сравнения чисел, занимающих целое слово, посредством вычитания — проблема, возникающая на многих моделях компьютеров; это основной довод в пользу включения операторов CMPA, ..., CMPX в систему команд компьютера MIX.)

7.
```
COMPARE STJ  9F          DEC1 1
1H      LDX  A,1         J1P  1B
        CMPX B,1       9H JMP  *  ▌
        JNE  9F
```

8. *Решение 1*, основанное на тождестве $\min(a, b) = \frac{1}{2}(a + b - |a - b|)$:

```
SOL1 LDA  A
     SRAX 5
     DIV  =2=
     STA  A1    a = 2a1 + a2
     STX  A2    |a2| ≤ 1
     LDA  B
     SRAX 5
     DIV  =2=
     STA  B1    b = 2b1 + b2
     STX  B2    |b2| ≤ 1
     LDA  A1
     SUB  B1    Переполнение невозможно
     STA  AB1   a1 − b1
     LDA  A2
     SUB  B2
     STA  AB2   a2 − b2
     SRAX 1
     ADD  AB1
     ENTX 1
     SLAX 5
     MUL  AB2
     STX  AB3   (a2 − b2) sign(a − b)
     LDA  A2
     ADD  B2
     SUB  AB3
     SRAX 5
     DIV  =2=
     ADD  A1
     ADD  B1    Переполнение невозможно
     SUB  AB1(1:5)
     STA  C  ▌
```

Решение 2, основанное на том, что индексирование может вызвать сложные взаимные обмены:

```
SOL2 LDA  A
     STA  C
     STA  TA
     LDA  B
     STA  TB
```

Теперь включите в программу приведенную ниже последовательность операторов k раз подряд, где $2^k > 10^{10}$:

```
     LDA  TA          LDA  TB          INC1 0,2
     SRAX 5           SRAX 5           INC1 0,2
     DIV  =2=         DIV  =2=         INC1 0,2
     STX  TEMP        STX  TEMP        LD3  TMIN,1
     LD1  TEMP        LD2  TEMP        LDA  0,3
     STA  TA          STA  TB          STA  C
```

(Эта программа просматривает разряды в двоичном представлении a и b справа налево, сохраняя знаковые разряды.) В конец текста программы поместите следующую таблицу:

```
     HLT
     CON  C   -1  -1
     CON  B    0  -1
     CON  B   +1  -1
     CON  A   -1   0
TMIN CON  C    0   0
     CON  B    1   0
     CON  A   -1   1
     CON  A    0   1
     CON  C    1   1
```

9. Вероятность равна сумме $\sum_j \binom{r+j-1}{j}(-1)^j\binom{N}{r+j}x^{r+j}$, которую получим методом включения и исключения (упр. 1.3.3–26). Это также можно записать в виде бета-распределения $r\binom{N}{r}\int_0^x t^{r-1}(1-t)^{N-r}\,dt$.

10. Рассортируйте содержимое ленты, а затем посчитайте. (Некоторые методы сортировки позволяют легко отбрасывать записи с повторяющимися ключами.)

11. Присвойте каждому человеку индивидуальный идентификационный номер, который должен быть указан на всех касающихся его карточках. Рассортируйте отдельно информационные и налоговые карточки, используя в качестве ключа идентификационный номер. Обозначим рассортированные налоговые карточки через $R_1, \ldots, R_N$, а их ключи — через $K_1 < \cdots < K_N$. (Не должно быть обнаружено ни одной пары карточек с повторяющимися ключами.) Добавьте новую $(N+1)$-ю запись с ключом ∞ и установите $i \leftarrow 1$. Затем для каждой записи из информационного файла проверьте, было ли о ней сообщено, следующим образом. Обозначим через K ключ обрабатываемой информационной карточки.

a) Если $K > K_i$, то увеличить i на 1 и повторить этот шаг.
b) Если $K < K_i$ или $K = K_i$, а соответствующая информация не отображена в налоговой карточке R_i, то сообщить об ошибке.

Постарайтесь проделать все это, не тратя впустую денег налогоплательщиков.

12. Один из способов состоит в том, чтобы присоединить к элементам матрицы $a_{i,j}$ ключи (j, i), лексикографически рассортировать элементы по этим ключам, после чего ключи

отбросить. (Подобная идея применима в *любом* случае, когда нужно переупорядочить информацию, если желаемый новый порядок выражается простой формулой.)

В частном случае, рассмотренном в этой задаче, можно воспользоваться сбалансированной двухпутевой сортировкой методом слияния, при которой ключи обрабатываются настолько просто, что их фактически не нужно явно записывать на ленту. Пусть задана матрица $n \times n$; тогда можно действовать следующим образом. Сначала запишем строки с нечетными номерами на ленту 1, а строки с четными номерами — на ленту 2:

Лента 1: $a_{11}\, a_{12} \ldots a_{1n}\, a_{31}\, a_{32} \ldots a_{3n}\, a_{51}\, a_{52} \ldots a_{5n} \ldots$
Лента 2: $a_{21}\, a_{22} \ldots a_{2n}\, a_{41}\, a_{42} \ldots a_{4n}\, a_{61}\, a_{62} \ldots a_{6n} \ldots$

Затем перемотаем ленты в начало и будем считывать с них информацию синхронно, чтобы получить

Лента 3: $a_{11}\, a_{21}\, a_{12}\, a_{22} \ldots a_{1n}\, a_{2n}\, a_{51}\, a_{61}\, a_{52}\, a_{62} \ldots a_{5n}\, a_{6n} \ldots$
Лента 4: $a_{31}\, a_{41}\, a_{32}\, a_{42} \ldots a_{3n}\, a_{4n}\, a_{71}\, a_{81}\, a_{72}\, a_{82} \ldots a_{7n}\, a_{8n} \ldots$

Перемотаем вторую пару лент в начало и вновь будем считывать с них данные синхронно:

Лента 1: $a_{11}\, a_{21}\, a_{31}\, a_{41}\, a_{12} \ldots a_{42} \ldots a_{4n}\, a_{9,1} \ldots$
Лента 2: $a_{51}\, a_{61}\, a_{71}\, a_{81}\, a_{52} \ldots a_{82} \ldots a_{8n}\, a_{13,1} \ldots$

И так до тех пор, пока после $\lceil \lg n \rceil$ просмотров не получится требуемая транспонированная матрица.

13. Один из методов состоит в том, чтобы добавить к записям различные случайные ключи, выполнить сортировку по этим ключам, а затем их убрать. (Ср. с упр. 12; аналогичный метод получения случайной *выборки* рассматривался в разделе 3.4.2.) Другой подход, требующий приблизительно столько же трудозатрат, но, по-видимому, в меньшей степени зависящий от качества генератора случайных чисел, состоит в том, чтобы присоединить к R_i случайное целое число в диапазоне $0 \le K_i \le N - i$, а затем переупорядочить записи, пользуясь методом из упр. 5.1.1–5.

14. Имея таблицу перекодировки, можно написать программу лексикографического сравнения, которая имитировала бы упорядочение, используемое на другом компьютере. Или же можно создать искусственные ключи, отличные от реальных литер, но дающие желаемое упорядочение. Последний метод имеет преимущество: такую операцию достаточно проделать один раз, однако потребуется больше памяти и нужно будет преобразовать весь ключ целиком. В первом методе результат сравнения часто можно определить, преобразовав всего одну-две буквы ключей; на последующих стадиях сортировки будут сравниваться почти равные ключи, поэтому в первом методе, видимо, выгоднее, прежде чем преобразовывать буквы, проверить, не равны ли они.

15. Заведите приблизительно 50 отдельных счетчиков и просмотрите весь файл один раз. Но если бы в условии задачи вместо слова "штат" стояло слово "город" и общее число городов было довольно велико, то следовало бы предпочесть сортировку по названиям городов.

16. Как и в упр. 15, решение зависит от объема задачи. Если количество идентификаторов для перекрестных ссылок не очень велико и их можно разместить в оперативной памяти, то лучше всего, пожалуй, воспользоваться алгоритмом обработки таблицы символов (см. гл. 6), в котором с каждым идентификатором связывается голова связного списка ссылок. Если же задача большая, необходимо создать файл записей, по одной на каждую ссылку, которая вносится в указатель, а затем рассортировать этот файл.

17. Присвойте всем карточкам "теневые ключи", чтобы требуемое упорядочение можно было выполнить с помощью обычной лексикографической сортировки по этим ключам.

Этот ключ должен быть внесен персоналом библиотеки в каждую карточку, когда она впервые поступает в систему. Можно, например, использовать следующий двухбуквенный код для разделения слов:

␣0	end of key;
␣1	end of cross-reference;
␣2	end of surname;
␣3	hyphen of multiple surname;
␣4	end of author name;
␣5	end of place name;
␣6	end of subject heading;
␣7	end of book title;
␣8	space between words.

Приведенный выше пример кодирования разделителей даст следующий результат (показаны только первые 25 литер):

```
ACCADEMIA␣8NAZIONALE␣8DEI
ACHTZEHNHUNDERTZWOLF␣8EIN
BIBLIOTHEQUE␣8D␣8HISTOIRE
BIBLIOTHEQUE␣8DES␣8CURIOS
BROWN␣2J␣8CROSBY␣4␣0
BROWN␣2JOHN␣4␣0
BROWN␣2JOHN␣4MATHEMATICIA
BROWN␣2JOHN␣4OF␣8BOSTON␣0
BROWN␣2JOHN␣41715␣0
BROWN␣2JOHN␣41715␣6␣0
BROWN␣2JOHN␣41761␣0
BROWN␣2JOHN␣41810␣0
BROWN␣3WILLIAMS␣2REGINALD
BROWN␣8AMERICA␣7␣0
BROWN␣8AND␣8DALLISONS␣8NE
BROWNJOHN␣2ALAN␣4␣0
DEN␣2VLADIMIR␣8EDUARDOVIC
DEN␣7␣0
DEN␣8LIEBEN␣8LANGEN␣8TAG␣
DIX␣2MORGAN␣41827␣0
DIX␣8HUIT␣8CENT␣8DOUZE␣80
DIX␣8NEUVIEME␣8SIECLE␣8FR
EIGHTEEN␣8FORTY␣8SEVEN␣8I
EIGHTEEN␣8TWELVE␣8OVERTUR
I␣8AM␣8A␣8MATHEMATICIAN␣7
I␣8B␣8M␣8JOURNAL␣8OF␣8RES
I␣8HA␣8EHAD␣7␣0
IA␣8A␣8LOVE␣8STORY␣7␣0
INTERNATIONAL␣8BUSINESS␣8
KHUWARIZMI␣2MUHAMMAD␣8IBN
LABOR␣7A␣8MAGAZINE␣8FOR␣8
LABOR␣8RESEARCH␣8ASSOCIAT
LABOUR␣1␣0
MACCALLS␣8COOKBOOK␣7␣0
MACCARTHY␣2JOHN␣41927␣0
MACHINE␣8INDEPENDENT␣8COM
MACMAHON␣2PERCY␣8ALEXANDE
MISTRESS␣8DALLOWAY␣7␣0
MISTRESS␣8OF␣8MISTRESSES␣
ROYAL␣8SOCIETY␣8OF␣8LONDO
SAINT␣8PETERSBURGER␣8ZEIT
SAINT␣8SAENS␣2CAMILLE␣418
SAINTE␣8MARIE␣2GASTON␣8P␣
SEMINUMERICAL␣8ALGORITHMS
UNCLE␣8TOMS␣8CABIN␣7␣0
UNITED␣8STATES␣8BUREAU␣80
VANDERMONDE␣2ALEXANDER␣8T
VANVALKENBURG␣2MAC␣8ELWYN
VONNEUMANN␣2JOHN␣41903␣0
WHOLE␣8ART␣8OF␣8LEGERDEMA
WHOS␣8AFRAID␣8OF␣8VIRGINI
WIJNGAARDEN␣2ADRIAAN␣8VAN
```

За таким вспомогательным ключом должны следовать данные из соответствующей карточки, чтобы можно было различать неодинаковые карточки с одинаковыми ключами (т. е. “Sir John” означает “John”). Обратите внимание, что “Saint-Saëns” — это имя с дефисом, а не составное имя. Год рождения al-Khuwārizmī* должен быть представлен в виде ␣40779 с ведущим нулем. (Данная схема будет хорошо работать до 9999 года, после чего мир окажется перед лицом очередного информационного кризиса.)

---

* Имя арабского ученого Аль-Хорезми в английской транскрипции. — *Прим. перев.*

Внимательно изучив этот пример, вы сможете решать аналогичные задачи с другими необычными типами упорядочения, которые требуют совместных усилий человека и компьютера.

18. Подготовьте два файла, из которых один содержит числа $(u^6 + v^6 + w^6) \bmod m$, а второй — числа $(z^6 - x^6 - y^6) \bmod m$, $u \le v \le w$, $x \le y \le z$, где m — длина слова в используемом компьютере. Рассортируйте оба файла и найдите в них общие элементы, которые нужно будет подвергнуть дальнейшему анализу. (Можно наложить дальнейшие ограничения на u, v, w, x, y, z с учетом того, что малые простые числа можно также сравнить по модулю.)

19. В общем случае для нахождения всех пар чисел $\{x_i, x_j\}$, таких, что $x_i + x_j = c$, при заданном c достаточно рассортировать файл так, чтобы $x_1 < x_2 < \cdots < x_N$. Присвоить $i \leftarrow 1$, $j \leftarrow N$ и затем повторять следующую процедуру до тех пор, пока не будет удовлетворено условие $j \le i$:

если $x_i + x_j = c$, то вывести $\{x_i, x_j\}$, установить $i \leftarrow i + 1$, $j \leftarrow j - 1$;
если $x_i + x_j < c$, то установить $i \leftarrow i + 1$;
если $x_i + x_j > c$, то установить $j \leftarrow j - 1$.

И наконец, если $j = i$ и $2x_i = c$, вывести $\{x_i, x_i\}$. Этот процесс аналогичен методу, описанному в упр. 18; по существу, мы подготовили два рассортированных файла, в одном из которых записаны $x_1, \ldots, x_N$, а в другом — $c - x_N, \ldots, c - x_1$, и отыскали в них дублирующиеся элементы. Но в данном случае второй файл не обязательно должен присутствовать в явном виде. При нечетном c возможен и другой подход — рассортировать файл по ключу (x нечетное $\Rightarrow x$, x четное $\Rightarrow c - x$).

Аналогичный алгоритм можно использовать и для поиска $\max\{x_i + x_j \mid x_i + x_j \le c\}$ или, скажем, для поиска при заданном t $\min\{x_i + y_j \mid x_i + y_j > t\}$ в двух рассортированных файлах — $x_1 \le \cdots \le x_m$ и $y_1 \le \cdots \le y_n$.

20. Вот некоторые из возможных решений. (а) Для каждой из 499 500 пар i, j, таких, что $1 \le i < j \le 1000$, установить $y_1 \leftarrow x_i \oplus x_j$, $y_2 \leftarrow y_1 \wedge (y_1 - 1)$, $y_3 \leftarrow y_2 \wedge (y_2 - 1)$, после чего вывести (x_i, x_j) в том и только в том случае, если $y_3 = 0$. Здесь символом "$\oplus$" обозначена операция "исключающее или", а символом "$\wedge$" — операция "поразрядное и". (б) Создать файл из 31 000 элементов, образовав для каждого исходного слова x_i по 31 элементу, а именно — само x_i и еще 30 слов, отличающихся от него значением в одном из разрядов. Затем рассортировать этот файл и найти повторяющиеся элементы. (в) Выполнить анализ, аналогично варианту (а), для

i) всех пар слов, старшие 10 разрядов которых совпадают;

ii) всех пар слов, средние 10 разрядов которых совпадают (но старшие 10 разрядов различаются);

iii) всех пар слов, младшие 10 разрядов которых совпадают (но ни старшие, ни средние не совпадают).

Этот метод предполагает выполнение трех процедур сортировки (каждая из которых — по одному из описанных 10-разрядных ключей). Если исходные слова представляют собой случайный набор, то ожидаемое число пар в каждом из трех случаев будет, по крайней мере, $499500/2^{10}$, что не превышает 500.

21. Прежде всего подготовьте файл из всех пятибуквенных английских слов. (Не забудьте о том, что такие слова могут образовываться и путем добавления суффиксов наподобие `-ED`, `-ER`, `-ERS`, `-S` к более коротким словам.) Затем рассортируйте буквы каждого слова α по возрастанию и обозначьте этот упорядоченный набор через α'. Наконец, рассортируйте множество пар (α', α), собрав таким образом вместе все анаграммы.

Эксперименты Кима Д. Гибсона (K. D. Gibson), проведенные в 1967 году, показали, что вторая по величине группа пятибуквенных анаграмм из общеупотребительных слов — это LEAST, SLATE, STALE, STEAL, TAELS, TALES, TEALS. Но если воспользоваться более обширным словарем, то в нее можно добавить слова ALETS (стальные наплечники), ASTEL (осколок), ATLES (намерения), LAETS (люди, занимающие промежуточное положение между рабами и свободными гражданами), LASET (горностай), LATES (нильский окунь), LEATS (канавы), SALET (средневековый шлем), SETAL (относящийся к остюкам), SLEAT (подстрекать), STELA (колонна, стела) и TESLA (единица измерения магнитной индукции). Если добавить сюда еще и устаревшие написания слов "settle" и "teasel" (SATEL, TASEL и TASLE), то получим в результате 22 слова из тех же букв, ни одно из которых не представляется в орфографическом словаре прописными литерами. Потратив еще немного времени, можно добавить в набор и слово "*tæsl*" из староанглийского языка, "*altes*" — из немецкого и "Madame de Staël" — из французского! Набор {LAPSE, LEAPS, PALES, PEALS, PLEAS, SALEP, SEPAL} также можно расширить, по крайней мере, до 14 слов, если всерьез взяться за специальные словари. [См. H. E. Dudeney, *300 Best Word Puzzles*, edited by Martin Gardner (New York: Chas. Scribner's Sons, 1968), Puzzles 190 and 194; Ross Eckler, *Making the Alphabet Dance* (N.Y.: St. Martin's Griffin, 1997), Fig. 46c.]

Первой и последней анаграммами среди найденных наборов из пятибуквенных анаграмм в английском языке длиной не менее трех элементов являются {ALBAS, BALAS, BALSA, BASAL} и {STRUT, STURT, TRUST}, если не принимать во внимание подходящих имен собственных. Если же снять это ограничение, то имена Alban, Balan, Laban и Nabal образуют набор, занявший теперь первое место {ALBAN, BALAN, BANAL, LABAN, NABAL, NABLA}. Самый впечатляющий пример длинных анаграмм в английском языке дают математические термины: {ALERTING, ALTERING, INTEGRAL, RELATING, TRIANGLE}. Сюда можно добавить и австралийскую рыбу TERAGLIN.

Можно ускорить процесс, вычисляя $f(\alpha) = (h(a_1) + h(a_2) + \cdots + h(a_5)) \bmod m$, где $a_1, \ldots, a_5$ — числовые коды букв слова α, а $(h(1), h(2), \ldots)$ представляют собой 26 случайно выбранных констант; здесь m — длина машинного слова. В результате сортировки файла $(f(\alpha), \alpha)$ все анаграммы будут собраны вместе; после этого для всех случаев $f(\alpha) = f(\beta)$ нужно убедиться в том, что найдена действительно анаграмма, т. е. $\alpha' = \beta'$. Значение $f(\alpha)$ вычисляется существенно быстрее, чем α', и этот метод позволяет избежать вычисления α' для большинства слов α из файла.

Замечание. Аналогичным методом можно воспользоваться, если нужно собрать все множества записей, имеющих равные многословные ключи $(a_1, \ldots, a_n)$. Предположим, что нам безразличен порядок записей, важно только, чтобы записи с равными ключами располагались подряд. В этом случае иногда удобно выполнить сортировку по однословному ключу $(a_1 x^{n-1} + a_2 x^{n-2} + \cdots + a_n) \bmod m$, где x — любая фиксированная величина, вместо того чтобы сортировать по исходному многословному ключу.

22. Найдите инварианты графов (т. е. функции, принимающие равные значения на изоморфных ориентированных графах) и выполните сортировку по этим инвариантам, чтобы отделить одну от другой группы "явно неизоморфных" графов. Далее приводятся примеры инвариантов. (а) Представьте вершину v_i в виде пары (a_i, b_i), где a_i — полустепень захода, а b_i — полустепень исхода вершины, после чего рассортируйте пары (a_i, b_i) в лексикографическом порядке. Полученный файл представляет собой инвариант изоморфных графов. (б) Представьте дугу из v_i к v_j в виде совокупности (a_i, b_i, a_j, b_j) и рассортируйте эти тетрады в лексикографическом порядке. (с) Разделите ориентированный граф на связанные компоненты (см. алгоритм 2.3.3E), определите инварианты каждой компоненты и любым способом рассортируйте эти компоненты в порядке инвариантов. (См. также ответ к упр. 21.)

В общем случае после сортировки ориентированных графов по инвариантам необходимо дополнительно проанализировать, являются ли графы с равными инвариантами в самом деле изоморфными. При выполнении этого анализа могут пригодиться и сформированные инварианты. В случае свободных деревьев можно найти "характеристические" или "канонические" инварианты, которые полностью характеризуют дерево, так что необходимость в дополнительном анализе исчезнет. [См. J. Hopcroft, R. E. Tarjan, *Complexity of Computer Computations* (New York: Plenum, 1972), 140–142.]

23. Один из способов выполнения этого упражнения — сформировать файл, содержащий все клики из трех человек, преобразовать его в файл, содержащий все клики из четырех человек, и т. д.; если клики не слишком велики, этот метод вполне подойдет. (С другой стороны, если есть клика размером n, то найдется, по крайней мере, $\binom{n}{k}$ клик размером k; так что этим методом не удастся получить требуемый результат, даже если n приблизительно равно 25 или более того.)

Если имеется файл, в котором перечислены все клики размером $(k-1)$ в виде $(a_1, \ldots, a_{k-1})$, где $a_1 < \cdots < a_{k-1}$, то клики размером k можно отыскать следующим образом: (i) сформировать новый файл, в который включить элементы $(b, c, a_1, \ldots, a_{k-2})$ для каждой пары клик размером $(k-1)$ в виде $(a_1, \ldots, a_{k-2}, b)$, $(a_1, \ldots, a_{k-2}, c)$, где $b < c$; (ii) рассортировать этот файл по первым двум компонентам; (iii) для каждого элемента $(b, c, a_1, \ldots, a_{k-2})$ этого нового файла (при том что пара (b, c) принадлежит исходному файлу знакомств) вывести в качестве результата клику $(a_1, \ldots, a_{k-2}, b, c)$ размером k.

24. (Это решение предложено Норманом Харди (N. Hardy) (1967).) Скопируем исходный файл; рассортируем одну копию по первым компонентам, а другую — по вторым. Теперь можно, просмотрев последовательно эти два файла, создать новый, в который включить все пары (x_i, x_{i+2}), $1 \le i \le N-2$, и найти (x_{N-1}, x_N). Пары $(N-1, x_{N-1})$ и (N, x_N) следует записать в еще один файл.

Далее процесс продолжается по индукции. Предположим, файл F содержит все пары (x_i, x_{i+t}), $1 \le i \le N-t$, в произвольном порядке, а файл G содержит все пары (i, x_i), $N-t < i \le N$, упорядоченные по вторым компонентам. Пусть H — копия файла F; рассортируем H по первому компоненту, а F — по второму. Просмотрим теперь F, G, и H и сформируем два новых файла F' и G' следующим образом. Если текущие записи файлов F, G, H равны соответственно (x, x'), (y, y'), (z, z'), то:

i) если $x' = z$, выведем (x, z') в F' и перейдем дальше в файлах F и H;

ii) если $x' = y'$, выведем $(y-t, x)$ в G' и перейдем дальше в файлах F и G;

iii) если $x' > y'$, перейдем дальше в файле G;

iv) если $x' > z$, перейдем дальше в файле H.

Когда дойдем до конца файла F, рассортируем G' по вторым компонентам и сольем с ним G; затем заменим t на $2t$, F на F', G на G'.

Таким образом, t принимает значения $2, 4, 8, \ldots$; при фиксированном t для сортировки необходимо выполнить $O(\log N)$ проходов. Следовательно, общее число проходов будет $O((\log N)^2)$. В конце концов выполнится $t \ge N$ и файл F опустеет; тогда останется только рассортировать G по *первым* компонентам.

25. (Идея решения принадлежит Д. Шанксу (D. Shanks).) Подготовьте два файла, один из которых содержит числа $a^{mn} \bmod p$, а другой — $ba^{-n} \bmod p$, $0 \le n < m$. Рассортируйте оба файла и найдите общий элемент.

Замечание. В результате время выполнения $\Theta(p)$ в худшем случае сократится до $\Theta(\sqrt{p} \log p)$. Возможно и дальнейшее повышение производительности; например, несложно за $\log p$ шагов проанализировать, является ли n четным, проверив, какое из равенств $b^{(p-1)/2} \bmod p = 1$ и $(p-1)$ имеет место. В общем случае, если f является любым делителем

$p-1$, а d — любым делителем $\gcd(f,n)$, можно аналогичным образом найти $(n/d) \bmod f$, отыскав значение $b^{(p-1)/f}$ в таблице длиной f/d. Если простыми делителями $p-1$ являются $q_1 \le q_2 \le \cdots \le q_t$, причем, если q_t — мало́, можно быстро вычислить n, выполняя поиск разрядов справа налево в представлении со смешанными основаниями $q_1, \ldots, q_t$. (Эта идея принадлежит Р. Л. Сильверу (R. L. Silver), 1964; см. также S. C. Pohlig, M. Hellman, *IEEE Transactions* **IT-24** (1978), 106–110.)

Джон М. Поллард (John M. Pollard) нашел довольно элегантный способ вычисления дискретных логарифмов, требующий порядка $O(\sqrt{p})$ операций $\bmod p$ при малых затратах памяти. Способ основан на теории случайных отображений (random mappings). См. *Math. Comp.* **32** (1978), 918–924, где он предложил еще один метод, базирующийся на числах $n_j = r^j \bmod p$, которые имеют только малые простые делители.

Асимптотически более быстрые методы обсуждаются в ответе к упр. 4.5.4–46.

РАЗДЕЛ 5.1.1

1. 205223000; 27354186.

2. $b_1 = (m-1) \bmod n$; $b_{j+1} = (b_j + m - 1) \bmod (n-j)$.

3. $\overline{a}_j = a_{n+1-j}$ ("отраженная" перестановка). Эту идею использовал О. Теркем (O. Terquem) [*Journ. de Math.* **3** (1838), 559–560] для доказательства того, что среднее число инверсий в случайной перестановке равно $\frac{1}{2}\binom{n}{2}$.

4. **C1.** Установить $x_0 \leftarrow 0$. (Если возможно, желательно на последующих циклах, когда $1 \le j \le n$, использовать для x_j ту же память, что и для b_j.)

C2. При $k = n, n-1, \ldots, 1$ (именно в этом порядке) выполнить следующее: установить $j \leftarrow 0$, затем ровно b_k раз установить $j \leftarrow x_j$, затем установить $x_k \leftarrow x_j$ и $x_j \leftarrow k$.

C3. Установить $j \leftarrow 0$.

C4. При $k = 1, 2, \ldots, n$ (именно в этом порядке) выполнить следующее: установить $a_k \leftarrow x_j$, затем установить $j \leftarrow x_j$. ▌

В упр. 5.2–12 показано, как экономно использовать память.

5. Пусть α — цепочка упорядоченных пар целых неотрицательных чисел $[m_1, n_1] \ldots [m_k, n_k]$; обозначим через $|\alpha| = k$ длину цепочки α, а через ϵ — пустую (длиной 0) цепочку. Рассмотрим двоичную операцию $\circ$, рекурсивно определенную на парах таких цепочек следующим образом:

$$\epsilon \circ \alpha = \alpha \circ \epsilon = \alpha;$$

$$([m,n]\alpha) \circ ([m',n']\beta) = \begin{cases} [m,n](\alpha \circ ([m'-m,n']\beta)), & \text{если } m \le m', \\ [m',n'](([m-m'-1,n]\alpha) \circ \beta), & \text{если } m > m'. \end{cases}$$

Из такого определения операции следует, что время, необходимое для вычисления $\alpha \circ \beta$, пропорционально $|\alpha \circ \beta| = |\alpha| + |\beta|$. Более того, можно показать, что операция $\circ$ ассоциативна и что $[b_1,1] \circ [b_2,2] \circ \cdots \circ [b_n,n] = [0,a_1][0,a_2]\ldots[0,a_n]$. Выражение в левой части можно вычислить за $\lceil \lg n \rceil$ проходов, комбинируя на каждом из них пары в цепочках, т. е. всего за $O(n \log n)$ шагов.

Пример. Начав с (2), зададимся целью вычислить $[2,1] \circ [3,2] \circ [6,3] \circ [4,4] \circ [0,5] \circ [2,6] \circ [2,7] \circ [1,8] \circ [0,9]$. После первого прохода это выражение преобразуется в $[2,1][1,2] \circ [4,4][1,3] \circ [0,5][2,6] \circ [1,8][0,7] \circ [0,9]$. Второй проход преобразует его в $[2,1][1,2][1,4][1,3] \circ [0,5][1,8][0,6][0,7] \circ [0,9]$. Третий проход приведет к $[0,5][1,1][0,8][0,2][0,6][0,4][0,7][0,3] \circ [0,9]$. После четвертого прохода получим (1).

Обоснование. Цепочка наподобие [4,4][1,3] представляет "␣␣␣␣4␣3␣$^\infty$", где "␣" означает пропуск; операция $\alpha \circ \beta$ вставляет пропуски и заполнения из β на место пропусков в α.

Обратите внимание, что, в отличие от упр. 2, получен алгоритм решения задачи Иосифа, который требует времени, пропорционального $O(n \log n)$, вместо $O(mn)$. Таким образом, мы частично ответили на вопрос, поставленный в упр. 1.3.2–22.

Другое решение этой задачи ($O(n \log n)$, которое потребует только оперативной памяти), очевидно, следует из методики сбалансированного дерева.

6. Начать с $b_1 = b_2 = \cdots = b_n = 0$. При $k = \lfloor \lg n \rfloor, \lfloor \lg n \rfloor - 1, \ldots, 0$ выполнить следующее: установить $x_s \leftarrow 0$ для $0 \le s \le n/2^{k+1}$; затем для $j = 1, 2, \ldots, n$ установить $r \leftarrow \lfloor a_j/2^k \rfloor \bmod 2$, $s \leftarrow \lfloor a_j/2^{k+1} \rfloor$ (это, по существу, выделение отдельных разрядов); если $r = 0$, установить $b_{a_j} \leftarrow b_{a_j} + x_s$, а если $r = 1$, установить $x_s \leftarrow x_s + 1$.

Другое решение будет продемонстрировано в упр. 5.2.4–21.

7. $B_j < j$ и $C_j \le n - j$, поскольку a_j имеет $j - 1$ элементов слева от него и $n - j$ элементов справа. Для того чтобы восстановить $a_1 a_2 \ldots a_n$ по $B_1 B_2 \ldots B_n$, начните с элемента 1; затем для $k = 2, \ldots, n$ добавляйте по 1 к каждому элементу, большему или равному $k - B_k$, и справа приписывайте элементы $k - B_k$ (см. метод 2 в разделе 1.2.5). Аналогичная процедура годится и для таблицы C_k. Альтернативный метод предполагает использование результатов следующего упражнения. [Таблица инверсий c_k была рассмотрена Родригесом (B. O. Rodrigues) в *J. de Math.* **4** (1839), 236–240; впервые таблица инверсий C_k появилась в книге Netto *Lehrbuch der Combinatorik* (1901), §5.]

8. $b' = C$, $c' = B$, $B' = c$, $C' = b$, поэтому каждой инверсии (a_i, a_j) таблицы $a_1 \ldots a_n$ соответствует инверсия (j, i) перестановки $a'_1 \ldots a'_n$. Справедливы и другие соотношения: (a) $c_j = j - 1$ тогда и только тогда, когда ($b_i > b_j$ для всех $i < j$); (b) $b_j = n - j$ тогда и только тогда, когда ($c_i > c_j$ для всех $i > j$); (c) $b_j = 0$ тогда и только тогда, когда ($c_i - i < c_j - j$ для всех $i > j$); (d) $c_j = 0$ тогда и только тогда, когда ($b_i + i < b_j + j$ для всех $i < j$); (e) $b_i \le b_{i+1}$ тогда и только тогда, когда $a'_i < a'_{i+1}$ и $c_i \ge c_{i+1}$; (f) $a_j = j + C_j - B_j$; $a'_j = j + b_j - c_j$.

9. Равенство $b_k = C_k = b'_k$ эквивалентно равенству $a_k = a'_k$.

10. $\sqrt{10}$. (Один из способов ввести систему координат для усеченного октаэдра — поставить в соответствие взаимным обменам соседних элементов 21, 43, 41, 31, 42, 32 векторы $(1,0,0)$, $(0,1,0)$, $\frac{1}{2}(1,1,\sqrt{2})$, $\frac{1}{2}(1,-1,\sqrt{2})$, $\frac{1}{2}(-1,1,\sqrt{2})$, $\frac{1}{2}(-1,-1,\sqrt{2})$. Сумма этих векторов равна $(1,1,2\sqrt{2})$ и представляет собой разность между вершинами 4321 и 1234.)

Существует решение, имеющее более симметричный вид — представить вершину π в *четырехмерном* пространстве следующим образом:

$$\sum \{\mathbf{e}_u - \mathbf{e}_v \mid (u, v) \text{ инверсия перестановки } \pi\},$$

где $\mathbf{e}_1 = (1,0,0,0)$, $\mathbf{e}_2 = (0,1,0,0)$, $\mathbf{e}_3 = (0,0,1,0)$, $\mathbf{e}_4 = (0,0,0,1)$. Таким образом, $1\,2\,3\,4 \leftrightarrow (0,0,0,0)$; $1\,2\,4\,3 \leftrightarrow (0,0,-1,1)$; $\ldots$; $4\,3\,2\,1 \leftrightarrow (-3,-1,1,3)$. Все точки лежат на трехмерном подпространстве $\{(w,x,y,z) \mid w + x + y + z = 0\}$; расстояние между соседними вершинами равно $\sqrt{2}$. Так же (см. ответ 8, (f)) можно представить $\pi = a_1\, a_2\, a_3\, a_4$ вектором (a'_1, a'_2, a'_3, a'_4), где $a'_1\, a'_2\, a'_3\, a'_4$ — обратная перестановка. (Такое 4-D представление усеченного октаэдра перестановками в качестве координат было рассмотрено вместе с обобщением на n-мерное пространство С. Говардом Хинтоном (C. Howard Hinton) в книге *The Fourth Dimension* (London, 1904), глава 10. Много лет спустя другие свойства были найдены Гилбодом (Guilbaud) и Розентилем (Rosenstiehl), которые назвали объект, изображенный на рис. 1, пермутаэдром (permutahedron); см. упр. 12.)

Копии усеченного октаэдра заполняют трехмерное пространство способом, который принято называть простейшим [см. H. Steinhaus, *Mathematical Snapshots* (Oxford, 1960), 200–203; C. S. Smith, *Scientific American* **190**, 1 (January, 1954), 58–64]. В книге V из *Collection* Паппюса (300 г. н. э.) усеченный октаэдр упоминается как одно из тринадцати особых

тел, исследованных Архимедом. Примером архимедова тела является непризматический многогранник, любая вершина которого симметрична по отношению к любой другой, а любая грань является правильным многоугольником, но не все они идентичны. Его можно найти, например, в книге W. W. Rouse Ball, *Mathematical Recreations and Essays*, revised by H. S. M. Coxeter (Macmillan, 1939), глава 5, или H. Martyn Cundy, A. P. Rollett *Mathematical Models* (Oxford, 1952), 94–109.

11. (a) Очевидно. (b) Постройте ориентированный граф с вершинами $\{1, 2, \ldots, n\}$ и дугами $x \to y$, если либо $x > y$ и $(x, y) \in E$, либо $x < y$ и $(y, x) \in \bar{E}$. Если этот граф не содержит ориентированных циклов, его вершины можно топологически рассортировать; полученное линейное упорядочение и есть требуемая перестановка. Если же ориентированные циклы присутствуют, то длина кратчайшего из них равна 3, потому что не бывает циклов длиной 1 или 2 и потому что более длинный цикл $a_1 \to a_2 \to a_3 \to a_4 \to \cdots \to a_1$ можно сократить (либо $a_1 \to a_3$, либо $a_3 \to a_1$). Но ориентированный цикл длиной 3 содержит две дуги либо из E, либо из $\bar{E}$, и, таким образом, доказано, что либо E, либо $\bar{E}$ не транзитивно.

12. [G. T. Guilbaud, P. Rosenstiehl, *Math. et Sciences Humaines* **4** (1963), 9–33.] Предположим, что $(a, b) \in \bar{E}$, $(b, c) \in \bar{E}$, $(a, c) \notin \bar{E}$. Тогда при некотором $k \geq 1$ имеем $a = x_0 > x_1 > \cdots > x_k = c$, где $(x_i, x_{i+1}) \in E(\pi_1) \cup E(\pi_2)$ при $0 \leq i < k$. Рассмотрим следующий контрпример при минимальном k. Поскольку $(a, b) \notin E(\pi_1)$ и $(b, c) \notin E(\pi_1)$, то $(a, c) \notin E(\pi_1)$; аналогично $(a, c) \notin E(\pi_2)$; следовательно, $k > 1$. Но если $x_1 > b$, то $(x_1, b) \in \bar{E}$ противоречило бы минимальности k, а из того, что $(x_1, b) \in E$, следовало бы $(a, b) \in E$. Аналогично, если $x_1 < b$, получится, что и $(b, x_1) \in \bar{E}$, и $(b, x_1) \in E$ невозможны.

13. Для любого фиксированного выбора $b_1, \ldots, b_{m-1}, b_{m+1}, \ldots, b_n$ в таблице инверсий в сумме $\sum_i b_i$ предполагается, что каждый остаток по модулю m встречается точно один раз по мере того, как b_m пробегает все возможные значения $0, 1, \ldots, m-1$.

14. Конструкция, предложенная в указании, переводит пары разбиений на различные части одно в другое во всех случаях, кроме двух: $j = k = p_k$ и $j = k = p_k - 1$. В этих особых случаях n равно соответственно $(2j-1)+\cdots+j = (3j^2-j)/2$ и $(2j)+\cdots+(j+1) = (3j^2+j)/2$ и существует единственное непарное разбиение на j частей. [Первоначальное доказательство Эйлера в *Novi Comment. Acad. Sci. Pet.* **5** (1754), 75–83, также очень интересно. С помощью простых преобразований он показал, что бесконечное произведение равно s_1, если мы определим s_n в виде степенного ряда $1 - z^{2n-1} - z^{3n-1}s_{n+1}$ для $n \geq 1$. Конечная версия бесконечной эйлеровой суммы рассмотрена Кнутом и Патерсоном (Paterson) в *Fibonacci Quarterly* **16** (1978), 198–212.]

15. Транспонируйте точечную диаграмму с тем, чтобы перейти от p к P. Производящая функция для P получается довольно легко, так как сначала можно выбрать любое число единиц (производящая функция $1/(1-z)$), затем независимо выбрать любое число двоек (производящая функция $1/(1-z^2)$), ... и, наконец, любое количество чисел n.

16. Коэффициент при $z^n q^m$ в первом тождестве равен количеству разбиений числа m на не более чем n частей. Во втором тождестве он равен количеству разбиений числа m на n различных неотрицательных частей, т. е. представляет собой сумму $m = p_1 + p_2 + \cdots + p_n$, где $p_1 > p_2 > \cdots > p_n \geq 0$. Это то же самое, что и разбиения $m - \binom{n}{2} = q_1 + q_2 + \cdots + q_n$, где $q_1 \geq q_2 \geq \cdots \geq q_n \geq 0$, так как можно установить взаимно однозначное соответствие $q_i = p_i - n + i$. [*Commentarii Academiæ Scientiarum Petropolitanæ* **13** (1741), 64–93.]

Замечание. Второе тождество получаем предельным переходом $n \to \infty$ вследствие q-номиальной теоремы (см. упр. 1.2.6–58). Аналогично и первое тождество получаем предельным переходом $r \to \infty$ в дуальной форме этой теоремы, доказанной в ответе к тому же упражнению.

Пусть $n!_q = \prod_{k=1}^{n}(1 + q + \cdots + q^{k-1})$ и $\exp_q(z) = \sum_{n=0}^{\infty} z^n/n!_q$. Из первого тождества следует, что $\exp_q(z)$ равно $1/\prod_{k=0}^{\infty}(1-q^k z(1-q))$ при $|q| < 1$; из второго следует, что это же

выражение равно $\prod_{k=0}^{\infty}(1+q^{-k}z(1-q^{-1}))$ при $|q|>1$. Полученное в результате тождество $\exp_q(z)\exp_{q^{-1}}(-z)=1$ эквивалентно формуле

$$\sum_{k=0}^{n}\frac{(-1)^k q^{k(k-1)/2}}{(1-q)\ldots(1-q^k)(1-q)\ldots(1-q^{n-k})}=\delta_{n0},\qquad \text{целые числа } n\ge 0,$$

которая является следствием из q-номиальной теоремы при $x=-1$.

17.

0 0 0 0	0 1 0 0	0 0 1 0	0 0 0 1
1 1 0 1	1 2 0 1	1 0 2 1	1 0 1 2
1 0 1 0	0 1 1 0	0 1 2 0	0 1 0 2
1 0 1 1	0 1 1 1	0 1 2 1	0 1 1 2
1 0 0 1	0 1 0 1	0 0 1 1	0 0 1 2
2 0 1 2	0 2 1 2	0 1 2 2	0 1 2 3

18. Пусть $q=1-p$. Сумму $\sum \Pr(\alpha)$ по всем случаям α инверсий можно вычислить суммированием по k, где $0\le k<n$ — точное число крайних битовых позиций, в которых соблюдается равенство между i и j и между X_i и X_j в инверсии, $X_i\oplus i>X_j\oplus j$ при $i<j$. Таким образом будет получена формула $\sum_{0\le k<n}2^k(p^2+q^2)^k(p^2 2^{n-k-1}2^{n-k-1}+2pq2^{n-k-1}(2^{n-k-1}-1))$. После суммирования и упрощения она преобразуется в $2^{n-1}(p(2-p)(2^n-(p^2+q^2)^n)/(2-p^2-q^2)+(p^2+q^2)^n-1)$.

19. Число инверсий равно

$$\sum_{0<i<j<n}(\lfloor mj/n\rfloor-\lfloor mi/n\rfloor-\lfloor m(j-i)/n\rfloor)=\sum_{0<i<j<n}[mj \bmod n<mi \bmod n]=$$
$$\sum_{0<r<n}\lfloor mr/n\rfloor(r-(n-r)-(n-r-1)),$$

что можно привести к виду $\frac14(n-1)(n-2)-\frac14 n\sigma(m,n,0)$. [См. *Crelle*, **198** (1957), 162–166.]

20. См. E. M. Wright, *J. London Math. Soc.* **40** (1965), 55–57; и J. Zolnowsky, *Discrete Math.* **9** (1974), 293–298.

Тождество Якоби можно быстро доказать следующим образом. Так как

$$\prod_{k=1}^{n}(1-u^kv^{k-1})=(-1)^n u^{\binom{n+1}{2}}v^{\binom{n}{2}}\prod_{k=1}^{n}(1-u^{-k}v^{1-k}),$$

вследствие q-номиальной теоремы, рассмотренной в упр. 1.2.6–58, при $q=uv$ имеем

$$\begin{aligned}\prod_{k=1}^{n}(1-u^kv^{k-1})(1-u^{k-1}v^k)&=(-1)^n u^{\binom{n+1}{2}}v^{\binom{n}{2}}\prod_{k=-n+1}^{n}(1-u^{k-1}v^k)\\&=(-1)^n u^{\binom{n+1}{2}}v^{\binom{n}{2}}\sum_j\binom{2n}{j}_{uv}(uv)^{\binom{k}{2}}(-u^{-n}v^{1-n})^k\\&=\sum_j\binom{2n}{n+j}_{uv}(-1)^j u^{\binom{j}{2}}v^{\binom{j+1}{2}}.\end{aligned}$$

Для фиксированных j после умножения обеих частей на $\prod_{k=1}^{n}(1-u^kv^k)=\prod_{k=1}^{n}(1-q^k)$ получим $\binom{2n}{n+j}_q\prod_{k=1}^{n}(1-q^k)=1+O(q^{n+1-|j|})$. При $n\to\infty$ отсюда следует тождество Якоби.

21. Интерпретируйте C_j как число элементов в стеке после j-го вывода. (Характеристики b_k и B_k таблиц перестановок в стеке рассмотрены в упр. 2.3.3–19.)

22. (a) Расставьте числа $\{1, 2, \ldots, n\}$ по кругу, как на циферблате часов, и установите указатель на 1. Затем для $j = n, n-1, \ldots, 1$ (именно в этом порядке) передвигайте указатель против часовой стрелки $h_j + 1$ шагов, удаляя из круга те числа, на которые он указывает, и назовите их a_j.

(b) Каждое i подсчитывается так часто, как "замыкается" последовательность $a_i\, a_{i+1} \ldots a_n$; это и будет число случаев, когда $a_j > a_{j+1}$ при $j \geq i$. Таким образом, каждое j при $a_j > a_{j+1}$ соответствует подсчитанным однократно индексам $1, \ldots, j$. [Guo-Niu Han, *Advances in Math.* **105** (1994), 28–29; тот же результат получен и Роулингсом (Rawlings) в контексте следующего упражнения.]

23. Предположим, например, что $n = 5$ и $a_1\, a_2\, a_3\, a_4\, a_5 = 3\,1\,4\,2\,5$. Тогда число холостых выстрелов перед каждым смертельным должно быть $2 + 5k_1$, $2 + 4k_2$, $1 + 3k_3$, $1 + 2k_2$, k_5, где k_j — некоторое неотрицательное целое. Обратите внимание, что дуальной перестановке $1\,4\,2\,5\,3$ соответствует h-таблица $0\,1\,1\,2\,2$ в обозначениях предыдущего упражнения. В общем случае вероятность серии $a_1\, a_2 \ldots a_n$ будет равна

$$\sum_{k_1,\ldots,k_n \geq 0} (q_1^{h_n + nk_1} p_1)(q_2^{h_{n-1} + (n-1)k_2} p_2) \ldots (q_n^{h_1 + k_n} p_n)$$
$$= \frac{1-q_1}{1-q_1^n} \frac{1-q_2}{1-q_2^{n-1}} \cdots \frac{1-q_n}{1-q_n^1} q_1^{h_n} q_2^{h_{n-1}} \ldots q_n^{h_1},$$

где $p_j = 1 - q_j$ — вероятность фатального исхода после предыдущих $j-1$ роковых выстрелов и $h_1\, h_2 \ldots h_n$ соответствует двойственной последовательности $a_1\, a_2 \ldots a_n$. В частности, при $p_1 = \cdots = p_n = p = 1 - q$ получим вероятность $q^{h_1 + \cdots + h_n}/G_n(q)$. Таким образом, будет порядок $n \ldots 2\,1$. [J. Treadway, D. Rawlings, *Math. Mag.* **67** (1994), 345–354; Роулингс обобщил этот результат для перестановок мультимножеств в *Int. J. Math. & Math. Sci.* **15** (1992), 291–312.]

24. Пусть $a_0 = 0$. Будем говорить, что *обобщенный спуск* возникает при $j < n$, если $a_j > t(a_{j+1})$. После вставки n между a_{j-1} и a_j появляется новый обобщенный спуск тогда и только тогда, когда $a_{j-1} \leq t(a_j) < n$. Предположим, что это произошло, когда j имело значение $j_1 > j_2 > \cdots > j_k > 0$; пусть другое значение j будет $j_n > j_{n-1} > \cdots > j_{k+1}$. Тогда $j_n = n$ и можно показать, что обобщенный индекс увеличивается на $n - k$, когда n вставлено именно перед a_{j_k}. [Особый случай, при котором $t(j) = j + d$ для некоторых $d \geq 0$, рассмотрен Д. Роулингсом, *J. Combinatorial Theory* **A31** (1981), 175–183; он обобщил его на перестановку мультимножеств в работе *Linear and Multilinear Algebra* **10** (1981), 253–260.

В этом упражнении определено $n!$ различных статистик перестановок, каждая из которых имеет производящую функцию $G_n(z)$, выведенную в (7) и (8). Можно определить и значительно больше таких статистик, обобщив формулировку задачи о русской рулетке следующим образом: после $j-1$ рокового выстрела новый круг начинается с участника $f_j(a_1, \ldots, a_{j-1})$, где f_j — произвольная функция, принимающая значения на $\{1, \ldots, n\} \setminus \{a_1, \ldots, a_{j-1}\}$. [См. Guo-Niu Han, *Calcul Denertien* (Thesis, Univ. Strasbourg, 1992), Part 1.3, §7.]

25. (a) Если $a_1 < a_n$, то $h(\alpha)$ имеет ровно столько же инверсий, сколько α, так как элементы в α_j теперь инвертировали x_j вместо a_n. Но, если $a_1 > a_n$, $h(\alpha)$ имеет на $n-1$ инверсий меньше, поскольку x_j теряет свою инверсию a_n и инверсию каждого элемента в α_j. Таким образом, если положить $x_n = a_n$ и рекурсивно переобозначить $x_1 \ldots x_{n-1} = f(h(\alpha))$, то перестановка $f(\alpha) = x_1 \ldots x_n$ будет обладать желаемыми свойствами. Получится $f(1\,9\,8\,2\,6\,3\,7\,4\,5) = 9\,1\,2\,6\,3\,8\,7\,4\,5$ и $f^{[-1]}(1\,9\,8\,2\,6\,3\,7\,4\,5) = 1\,9\,2\,6\,8\,7\,3\,4\,5$.

(b) Здесь ключевыми являются соотношения $\mathrm{inv}(\alpha) = \mathrm{inv}(\alpha^-)$ и $\mathrm{ind}(\alpha^-) = \mathrm{ind}(f(\alpha)^-)$, если α^- есть инверсия α. Таким образом, если $\alpha_1 = \alpha^-$, $\alpha_2 = f(\alpha_1)$, $\alpha_3 = \alpha_2^-$, $\alpha_4 = f^{[-1]}(\alpha_3)$ и $\alpha_5 = \alpha_4^-$, получим

$$\operatorname{inv}(\alpha_5) = \operatorname{inv}(\alpha_4) = \operatorname{ind}(\alpha_3) = \operatorname{ind}(\alpha_2^-) = \operatorname{ind}(\alpha_1^-) = \operatorname{ind}(\alpha);$$
$$\operatorname{ind}(\alpha_5) = \operatorname{ind}(\alpha_4^-) = \operatorname{ind}(\alpha_3^-) = \operatorname{ind}(\alpha_2) = \operatorname{inv}(\alpha_1) = \operatorname{inv}(\alpha).$$

[*Math. Nachrichten* **83** (1978), 143–159.]

26. (Решение предложено Дороном Зильбергером (Doron Zeilberger).) Среднее от $\operatorname{inv}(\alpha)$ $\operatorname{ind}(\alpha)$ равно

$$\frac{1}{n!}\sum_{\alpha}\sum_{1\le j<k\le n}\sum_{1\le l<n}[a_j > a_k]\; l\; [a_l > a_{l+1}],$$

что представляет собой полином от n степени, меньшей или равной 4. Вычисление этой суммы для $1 \le n \le 5$ дает соответственно значения 0, $\frac{1}{2}$, $\frac{6}{2}$, $\frac{21}{2}$, $\frac{55}{2}$; таким образом, полином имеет вид $\frac{1}{8}n(n-1) + \frac{1}{16}n^2(n-1)^2$. После вычитания $\operatorname{mean}(g_n)^2$ и деления на $\operatorname{var}(g_n)$ получим ответ: $9/(2n+5)$ для $n \ge 2$, если исходить из (12) и (13).

27. Рассматривая $q_n \ldots q_2\, q_1$ как перестановку мультимножества, получим $\operatorname{inv}(a_1\, a_2 \ldots a_n) = \operatorname{inv}(q_n \ldots q_2\, q_1)$ (см. раздел 5.1.2). Отсюда, используя обозначения из ответа к упр. 16 и результаты упр. 5.1.2–16, получим

$$\begin{aligned}
\sum_n \frac{H_n(w,z)}{(1-z)\ldots(1-z^n)} &= \sum_{a_1\ldots a_n} w^{\operatorname{inv}(a_1\ldots a_n)}\, z^{\operatorname{ind}(a_1\ldots a_n)} \sum_{p_1\ge\cdots\ge p_n\ge 0} z^{p_1+\cdots+p_n} \\
&= \sum_{q_1,q_2,\ldots,q_n\ge 0} w^{\operatorname{inv}(q_n\ldots q_2\, q_1)} z^{q_1+q_2+\cdots+q_n} \\
&= \sum_{k_0+k_1+k_2+\cdots=n} \binom{n}{k_0, k_1, k_2, \ldots}_w z^{k_1+2k_2+\cdots} \\
&= n!_w\, [u^n] \sum_{k_0,k_1,k_2,\ldots} \prod_{j=0}^{\in} fty \frac{(z^j u)^{k_j}}{k_j!_w} \\
&= n!_w\, [u^n] \prod_{j=0}^{\infty} \exp_w(z^j u) \\
&= n!_w\, [u^n] \prod_{j=0}^{\infty}\prod_{k=0}^{\infty} \frac{1}{1 - z^j w^k u(1-w)}.
\end{aligned}$$

В результате имеем элегантное тождество

$$\prod_{j,k\ge 0} \frac{1}{1 - w^j z^k u} = \sum_{n\ge 0} \frac{H_n(w,z)\, u^n}{(1-w)(1-w^2)\ldots(1-w^n)(1-z)(1-z^2)\ldots(1-z^n)},$$

из которого и появляется требуемая производящая функция $H_n(w,z) = \sum_\alpha w^{\operatorname{ind}(\alpha^-)} z^{\operatorname{ind}(\alpha)}$ (последняя получена Д. П. Розелем (D. P. Roselle) в *Proc. Amer. Math. Soc.* **45** (1974), 144–150). В упр. 25 показано, что та же функция двух переменных подсчитывает индексы и инверсии. Приведенное здесь доказательство принадлежит Гарсиа (Garsia) и Гесселю (Gessel) [см. *Advances in Math.* **31** (1979), 288–305], которые стремились получить существенно более общий результат.

Если в упр. 4.7–27 положить $m = \infty$, то придем к рекуррентному соотношению

$$H_n(w,z) = \sum_{k=1}^{n} \binom{n}{k}_w z^{n-k} \left(\prod_{j=1}^{k-1}(1 - z^{n-j})\right) H_{n-k}(w,z).$$

28. Взаимная замена двух соответствующих элементов изменяет значение суммарного смещения на величину 0 или ± 2; следовательно, суммарное смещение $\operatorname{td}(a_1\, a_2 \ldots a_n) \le 2\operatorname{inv}(a_1\, a_2 \ldots a_n)$.

Можно также доказать, что $\mathrm{td}(a_1\,a_2\ldots a_n) \geq \mathrm{inv}(a_1\,a_2\ldots a_n)$. Предположим, что j — наименьший элемент, который находится не на своем месте, и $a_k = j$. Пусть l будет максимальным, причем $l < k$ и $a_l \geq k$. Взаимный обмен a_l и a_k уменьшает количество инверсий на $2(k-l)-1$, а суммарное смещение — на $2(k-l)$. Таким образом, если для сортировки данной перестановки $a_1\,a_2\ldots a_n$ требуется m повторений этого алгоритма, получим $\mathrm{td}(a_1\,a_2\ldots a_n) = \mathrm{inv}(a_1\,a_2\ldots a_n) + m$.

Замечание. Среднее суммарное смещение случайной перестановки равно $(n^2-1)/3$ (см. упр. 5.2.1–7). Производящая функция для среднего суммарного смещения не имеет простой формы.

29. Можно получить π как произведение $\mathrm{inv}(\pi)$ операций транспонирования τ_j, где операция τ_j меняет местами j и $j+1$. Например, путь $1234 \to 1324 \to 1342 \to 3142$ на рис. 1 соответствует τ_2, затем — τ_3, затем — τ_1; следовательно, $3\,1\,4\,2 = \tau_1\tau_3\tau_2$. Таким образом, можно получить $\pi\pi'$ из π', выполнив $\mathrm{inv}(\pi)$ операций транспонирования, каждая из которых изменит количество инверсий на ± 1. Отсюда следует, что $\mathrm{inv}(\pi\pi') \leq \mathrm{inv}(\pi) + \mathrm{inv}(\pi')$. Если имеется равенство, каждое транспонирование добавляет одну инверсию, т. е. $E(\pi\pi') \supseteq E(\pi')$.

Значит, если $E(\pi\pi') \supseteq E(\pi')$, нам нужно показать, что некоторая последовательность из $|E(\pi\pi')| - |E(\pi')| = \mathrm{inv}(\pi\pi') - \mathrm{inv}(\pi')$ операций транспонирования преобразует π' в $\pi\pi'$. Данные операции транспонирования определяют π. Этим доказывается справедливость соотношения $\mathrm{inv}(\pi) \leq \mathrm{inv}(\pi\pi') - \mathrm{inv}(\pi')$; следовательно, должно соблюдаться равенство. Предположим, что $\pi' = 3\,1\,4\,5\,9\,2\,6\,8\,7$ и $E(\pi\pi') \supseteq E(\pi')$. Если $E(\pi\pi')$ не содержит $(4,1)$ или $(5,4)$, или $(9,5)$, или $(6,2)$, или $(8,6)$, то $\pi\pi'$ должно быть равно π'. В противном случае $E(\pi\pi')$ содержит одно из них, скажем, $(9,5)$; тогда $E(\pi\pi')$ включает $E(\tau_4\pi') = E(3\,1\,4\,9\,5\,2\,6\,8\,7)$. Подобным образом по индукции можно доказать справедливость указанного в условиях соотношения для $|E(\pi\pi')| - |E(\pi')|$.

РАЗДЕЛ 5.1.2

1. Не верно, поскольку оставлен без внимания один важный нюанс. Тот, кто ответил "верно", возможно, забыл (или не знает) данное в разделе 4.6.3 определение операции $M_1 \cup M_2$, согласно которому $M_1 \cup M_2$ является множеством только в том случае, если M_1 и M_2 также являются множествами. На самом деле $\alpha \top \beta$ — перестановка $M_1 \uplus M_2$.

2. $b\ c\ a\ d\ d\ a\ d\ a\ d\ b$.

3. Разумеется, нет, поскольку может быть и $\alpha = \beta$. (Однако теорема о единственности разложения показывает, что такие случаи встречаются не часто.)

4. $(d) \top (b\ c\ d) \top (b\ b\ c\ a\ d) \top (b\ a\ b\ c\ d) \top (d)$.

5. Число вхождений пары $\ldots xx \ldots$ равно числу столбцов $\begin{smallmatrix} x \\ x \end{smallmatrix}$ либо на единицу меньше. Если x — наименьший элемент, то эти числа равны тогда и только тогда, когда x — не первый элемент перестановки.

6. Несложно подсчитать соответствующее число двухстрочных массивов: $\binom{m}{k}\binom{n}{k}$.

7. Используя п. (а) теоремы B, выведем требуемые соотношения аналогично тому, как было выведено соотношение (20):

$$\binom{A-1}{A-k-m-1}\binom{B}{m}\binom{C}{k}\binom{B+k}{B-l}\binom{C-k}{l};$$

$$\binom{A-1}{A-k-m}\binom{B}{m}\binom{C}{k}\binom{B+k-1}{B-l-1}\binom{C-k}{l};$$

$$\binom{A-1}{A-k-m}\binom{B}{m}\binom{C}{k}\binom{B+k-1}{B-l}\binom{C-k}{l}.$$

8. Полное разложение на простые множители имеет вид $(d) \top (b\ c\ d) \top (b) \top (a\ d\ b\ c) \top (a\ b) \top (b\ c\ d) \top (d)$; оно единственно, поскольку никакие два соседние сомножители не коммутируют. Таким образом, имеется восемь решений с $\alpha = \epsilon$, (d), $(d) \top (b\ c\ d)$, ...

10. В общем случае утверждение ложно, но в некоторых интересных частных случаях — истинно. При любом заданном линейном упорядочении простых перестановок существует, по крайней мере, одно разложение указанного вида, потому что, если условие нарушается, можно сделать взаимную замену, которая сократит число "инверсий" в разложении. Поэтому условие может не выполняться только потому, что некоторые перестановки имеют не одно такое разложение.

Пусть $\rho \sim \sigma$ означает, что ρ коммутирует с σ. Следующее условие необходимо и достаточно для единственности разложения в указанном смысле:

$$\rho \sim \sigma \sim \tau \qquad \text{и} \qquad \rho \prec \sigma \prec \tau \qquad \text{влечет за собой} \qquad \rho \sim \tau.$$

Доказательство. Если бы $\rho \sim \sigma \sim \tau$, $\rho \prec \sigma \prec \tau$ и $\rho \not\sim \tau$, то существовало бы два разложения $\sigma \top \tau \top \rho = \tau \top \rho \top \sigma$; значит, условие необходимо. Обратно, для того чтобы показать, что оно достаточно для единственности, предположим, что $\rho_1 \top \cdots \top \rho_n = \sigma_1 \top \cdots \top \sigma_n$ — два различных разложения, удовлетворяющих условию. Можно считать, что $\sigma_1 \prec \rho_1$ и, следовательно, $\sigma_1 = \rho_k$ для некоторых $k > 1$; далее, $\sigma_1 \sim \rho_j$ при $1 \le j < k$. Поскольку $\rho_{k-1} \sim \sigma_1 = \rho_k$, то $\rho_{k-1} \prec \sigma_1$; следовательно, $k > 2$. Пусть j таково, что $\sigma_1 \prec \rho_j$ и $\rho_i \prec \sigma_1$ при $j < i < k$. Тогда из $\rho_{j+1} \sim \sigma_1 \sim \rho_j$ и $\rho_{j+1} \prec \sigma_1 \prec \rho_j$ следует, что $\rho_{j+1} \sim \rho_j$; отсюда получается $\rho_j \prec \rho_{j+1}$, а это — противоречие.

Таким образом, если на множестве простых перестановок S задано отношение порядка, удовлетворяющее указанному выше условию, и если известно, что все простые множители перестановки π принадлежат S, то можно заключить, что π имеет единственное разложение указанного типа. Такое условие выполняется, например, когда S представляет собой множество циклов в (29).

Но множество *всех* простых перестановок нельзя упорядочить таким образом. Если, скажем, $(a\ b) \prec (d\ e)$, то придется предположить, что

$$(a\ b) \prec (d\ e) \succ (b\ c) \prec (e\ a) \succ (c\ d) \prec (a\ b) \succ (d\ e),$$

а это — противоречие. (См. также следующее упражнение.)

11. Наша цель — показать, что если $p(1) \ldots p(t)$ есть перестановка множества $\{1, \ldots, t\}$, то перестановка $x_{p(1)} \ldots x_{p(t)}$ топологически рассортирована тогда и только тогда, когда $\sigma_{p(1)} \top \cdots \top \sigma_{p(t)} = \sigma_1 \top \cdots \top \sigma_t$, и что если $x_{p(1)} \ldots x_{p(t)}$ и $x_{q(1)} \ldots x_{q(t)}$ — различные топологические сортировки, то $\sigma_{p(j)} \neq \sigma_{q(j)}$ при некотором j. Первое свойство следует из того, что $x_{p(1)}$ может быть первым в топологической сортировке тогда и только тогда, когда $\sigma_{p(1)}$ коммутирует с $\sigma_{p(1)-1}, \ldots, \sigma_1$ (но отлично от него), а из этого условия следует, что $\sigma_{p(2)} \top \cdots \top \sigma_{p(t)} = \sigma_1 \top \cdots \top \sigma_{p(1)-1} \top \sigma_{p(1)+1} \top \cdots \top \sigma_t$ и можно использовать индукцию. Второе свойство вытекает из того, что если j — наименьший индекс, при котором $p(j) \neq q(j)$ (пусть для определенности $p(j) < q(j)$), и справедливо $x_{p(j)} \not\prec x_{q(j)}$ по определению топологической сортировки, то $\sigma_{p(j)}$ не имеет общих букв с $\sigma_{q(j)}$.

Для того чтобы получить произвольное частичное упорядочение, положим, что цикл σ_k состоит из всех упорядоченных пар (i, j), таких, что $x_i \prec x_j$ и либо $i = k$, либо $j = k$; эти упорядоченные пары входят в цикл в произвольном порядке в качестве отдельных элементов цикла. Так, для частичного упорядочения $x_1 \prec x_2$, $x_3 \prec x_4$, $x_1 \prec x_4$ циклы были бы следующими: $\sigma_1 = ((1,2)(1,4))$, $\sigma_2 = ((1,2))$, $\sigma_3 = ((3,4))$, $\sigma_4 = ((1,4)(3,4))$.

12. Никаких других циклов образовать нельзя хотя бы потому, что исходная перестановка не содержит столбцов $\binom{a}{c}$. Если $(a\ b\ c\ d)$ встречается s раз, то цикл $(a\ b)$ должен встретиться $A - r - s$ раз, поскольку имеется всего $A - r$ столбцов $\binom{a}{b}$ и только два вида циклов вносят вклад в эти столбцы.

13. Используя двухстрочное представление, сначала поместите $A - t$ столбцов вида $\binom{d}{a}$, затем — оставшиеся t букв a во вторую строку и буквы b, а также все остальные буквы.

14. Поскольку в двухстрочном представлении перестановки π^- элементы под любой буквой всегда располагаются в порядке неубывания, то не всегда выполняется равенство $(\pi^-)^- = \pi$, но всегда справедливо $((\pi^-)^-)^- = \pi^-$. При любых α и β имеет место тождество

$$(\alpha \top \beta)^- = ((\alpha^- \top \beta^-)^-)^-$$

(см. упр. 5–2).

Для данного мультимножества, все различные буквы которого суть $x_1 < \cdots < x_m$, можно охарактеризовать все перестановки, обратные самим себе, приняв во внимание, что каждая из них имеет единственное разложение на простые множители вида $\beta_1 \top \cdots \top \beta_m$, где β_j состоит из 0 или более простых множителей $(x_j) \top \cdots \top (x_j) \top (x_j x_{k_1}) \top \cdots \top (x_j x_{k_t})$, $j < k_1 \le \cdots \le k_t$. Например, $(a) \top (a\ b) \top (a\ b) \top (b\ c) \top (c)$ — перестановка, обратная самой себе. Следовательно, число обратных самим себе перестановок мультимножества $\{m \cdot a,\ n \cdot b\}$ равно $\min(m, n) + 1$, а соответствующее число для $(l \cdot a,\ m \cdot b,\ n \cdot c)$ есть число решений системы неравенств $x + y \le l$, $x + z \le m$, $y + z \le n$ в неотрицательных целых числах x, y, z. Число обратных самим себе перестановок *множества* рассматривается в разделе 5.1.4.

Количество перестановок мультимножества $(n_1 \cdot x_1, \ldots, n_m \cdot x_m)$, имеющих в своем двухстрочном представлении n_{ij} вхождений столбца $\binom{x_i}{x_j}$, есть $\prod_i n_i! / \prod_{i,j} n_{ij}!$. Столько же существует перестановок, в двухстрочной нотации которых содержится n_{ij} вхождений $\binom{x_j}{x_i}$. Следовательно, должен существовать другой, лучший способ определения обратной перестановки мультимножества. Например, если разложение на простые множители мультимножества π есть $\sigma_1 \top \sigma_2 \top \cdots \top \sigma_t$, как в теореме C, можно определить $\pi^- = \sigma_t^- \top \cdots \top \sigma_2^- \top \sigma_1^-$, где $(x_1 \ldots x_n)^- = (x_n \ldots x_1)$.

Доминик Фоата (Dominique Foata) и Гуо-Ню Хан (Guo-Niu Han) пришли к выводу, что было бы желательно определить обращение таким образом, чтобы π и π^- имели равное число обращений, поскольку производящая функция для обращений, дающих числа n_{ij}, есть $\prod_i n_i!_z / \prod_{i,j} n_{ij}!_z$ (см. упр. 16). Однако, как нам кажется, не существует естественного способа определить инволюцию, обладающую таким свойством.

15. См. теорему 2.3.4.2D. После удаления одной дуги ориентированного графа должно оставаться ориентированное дерево.

16. Если $x_1 < x_2 < \cdots$, то элементы таблицы инверсий для разных x_j должны иметь вид $b_{j1} \le \cdots \le b_{jn_j}$, где b_{jn_j} (число инверсий для крайнего справа элемента x_j) не больше, чем $n_{j+1} + n_{j+2} + \cdots$. Таким образом, производящая функция для j-й части таблицы инверсий есть производящая функция для разбиений не более чем на n_j частей, причем никакая часть не превосходит $n_{j+1} + n_{j+2} + \cdots$. Производящая функция для разбиений не более чем на m частей, где никакая часть не превосходит n, равна z-номиальному коэффициенту $\binom{m+n}{m}_z$, что довольно просто доказывается по индукции; это также можно доказать с помощью одного из остроумных умозаключений, принадлежащих Ф. Франклину (F. Franklin) [см. *Amer. J. Math.* **5** (1882), 268–269; см. также Pólya, Alexanderson, *Elemente der Mathematik* **26** (1971), 102–109]. Перемножив производящие функции при $j = 1, 2, \ldots$, можно получить искомую формулу для обращения перестановок мультимножеств, которую опубликовал Мак-Магон в *Proc. London Math. Soc.* (2) **15** (1916), 314–321.

17. Пусть $h_n(z) = (n!_z)/n!$, тогда желаемая вероятностная производящая функция имеет вид $g(z) = h_n(z)/h_{n_1}(z)h_{n_2}(z)\cdots$. Среднее от $h_n(z)$ равно $\frac{1}{2}\binom{n}{2}$ в соответствии с соотношением 5.1.1–(12), отсюда среднее значение для g равно

$$\frac{1}{2}\left(\binom{n}{2} - \binom{n_1}{2} - \binom{n_2}{2} - \cdots\right) = \frac{1}{4}(n^2 - n_1^2 - n_2^2 - \cdots) = \frac{1}{2}\sum_{i<j} n_i n_j.$$

Аналогично дисперсия равна

$$\tfrac{1}{72}(n(n-1)(2n+5)-n_1(n_1-1)(2n_1+5)-\cdots)$$
$$=\tfrac{1}{36}(n^3-n_1^3-n_2^3-\cdots)+\tfrac{1}{24}(n^2-n_1^2-n_2^2-\cdots).$$

18. Да, верно. Построения из упр. 5.1.1–25 очевидным образом распространяются и на этот случай. Можно также обобщить доказательство, которое приведено в разделе 5.1.1–(14), построив взаимно однозначное соответствие между совокупностями m элементов $(q_1,\ldots,q_m)$, где q_j — мультимножество, которое содержит n_j неотрицательных целых чисел, с одной стороны, и упорядоченными парами совокупностей n элементов $((a_1,\ldots,a_n),(p_1,\ldots,p_n))$, где $a_1\ldots a_n$ — перестановка мультимножества $\{n_1\cdot 1,\ldots,n_m\cdot m\}$ и $p_1\ge\cdots\ge p_n\ge 0$, с другой стороны. Это соответствие, как и прежде, определяется путем назначения всем элементам q_j нижнего индекса j; оно удовлетворяет условию

$$\Sigma(q_1)+\cdots+\Sigma(q_m)=\operatorname{ind}(a_1\ldots a_n)+(p_1+\cdots+p_n),$$

где $\Sigma(q_j)$ означает сумму элементов мультимножества q_j. [Дальнейшее обобщение методики, использованной в этом доказательстве и выводе соотношения 5.1.3–(8), приводится в работе D. E. Knuth, *Math. Comp.* **24** (1970), 955–961. См. также исчерпывающее исследование Richard P. Stanley, *Memoirs Amer. Math. Soc.* **119** (1972).]

19. (a) Пусть $S=\{\sigma\mid\sigma$ — простая перестановка, σ — левый множитель в разложении $\pi\}$. Если S состоит из k элементов, то левые множители λ в разложении перестановки π такие, что $\mu(\lambda)\ne 0$ есть не что иное, как ровно 2^k соединительных произведений подмножеств множества S (см. доказательство теоремы C); следовательно, $\sum\mu(\lambda)=\prod_{\sigma\in S}(1+\mu(\sigma))=0$, следовательно, $\mu(\sigma)=-1$ и S непусто. (b) Ясно, что $\epsilon(i_1\ldots i_n)=\mu(\pi)=0$, если $i_j=i_k$ при некотором $j\ne k$. В противном случае $\epsilon(i_1\ldots i_n)=(-1)^r$, где $i_1\ldots i_n$ имеет r инверсий; это ровно $(-1)^s$, где s — число четных циклов перестановки $i_1\ldots i_n$, что, в свою очередь, равно $(-1)^{n+t}$, где t — число циклов перестановки $i_1\ldots i_n$.

20. (a) Соотношение, очевидно, следует из определения соединительного произведения. (b) По определению

$$\det(b_{ij})=\sum_{1\le i_1,\ldots,i_m\le m}\epsilon(i_1\ldots i_m)\,b_{1i_1}\ldots b_{mi_m}.$$

Полагая $b_{ij}=\delta_{ij}-a_{ij}x_j$ и применяя результат упр. 19, (b), получим

$$\sum_{n\ge 0}\ \sum_{1\le i_1,\ldots,i_n\le m}x_{i_1}\ldots x_{i_n}\,\mu(x_{i_1}\ldots x_{i_n})\nu(x_{i_1}\ldots x_{i_n}),$$

поскольку $\mu(\pi)$, как правило, равно нулю.

(c) Используйте результат упр. 19, (a) и покажите, что $D\top G=1$, если рассматривать произведения всевозможных элементов x как перестановки некоммутативных переменных, учитывая естественное алгебраическое соглашение $(\alpha+\beta)\top\pi=\alpha\top\pi+\beta\top\pi$.

Сжатое толкование этого комбинаторного доказательства и похожие доказательства других важных теорем даны Д. Зильбергером [D. Zeilberger, *Discrete Math.* **56** (1985), 61–72].

21. $\prod_{k=1}^m\binom{n_k+\cdots+n_{k-d}}{n_k}$, если положить $n_k=0$ при $k\le 0$, поскольку существует $\binom{n_m+\cdots+n_{m-d}}{n_m}$ способов вставить элементы m в такую перестановку мультимножества $\{n_1\cdot 1,\ldots,n_{m-1}\cdot(m-1)\}$.

22. (a) Перестановка слева направо $l(\pi)$ существует в $P_0(0^k1^{n_1}\ldots t^{n_t})$ для некоторого k; но вместо перестановки $l(\pi)$ рассмотрим ее двухстрочное представление, в котором 0 размещен последним, а не первым в верхней строке. Число k элементов 0 в $l(\pi)$ и $r(\pi)$ равно числу столбцов $\binom{j}{k}$ в двухстрочном представлении π, для которых $j\le t<k$; это также

и число столбцов, для которых $k \le t < j$. Можно легко воспроизвести π из двухстрочных представлений $l(\pi)$ и $r(\pi)$, поскольку каждый столбец $\binom{j}{k}$ с $j, k \le t$ включен в $l(\pi)$, каждый столбец с $t < j, k$ включен в $r(\pi)$, а оставшиеся столбцы получены путем слияния $\binom{j}{0}$ или $\binom{0}{k}$ из $l(\pi)$ с $\binom{0}{k}$ или $\binom{j}{0}$ из $r(\pi)$ слева направо.

(b) Пусть π — начальная форма перестановки и пусть σ — любая перестановка из $P_0(0^{n_0}1^{n_1}\dots m^{n_m})$. Сформируйте λ следующим образом: удалите первые n_0 элементов из σ; затем замените 0 элементами x, имеющими подстрочные индексы из первых n_0 элементов π; замените другие элементы элементами y, имеющими подстрочные индексы из оставшихся ненулевых элементов из π. Сформируйте также ρ следующим образом: удалите элементы 0 из σ и замените n_j появлений j на x_j или y_j соответственно тому, имеют ли столбцы $\binom{j}{k}$ из π $k = 0$ или $k \ne 0$, в порядке слева направо. Например, если $\pi = \begin{smallmatrix}0000001111122223333\\2313130231010203201 0\end{smallmatrix}$, а $\sigma = \begin{smallmatrix}0000001111122223333\\3231320110320130020 1\end{smallmatrix}$, то получим $\lambda = x_2y_2y_3x_3y_1y_1x_1y_2y_3x_3x_1y_2x_3y_1$ и $\rho = y_3y_2y_3x_1x_3x_2y_1y_1y_3y_2y_1x_3x_2x_1$. И обратно, можно воссоздать π и σ из λ и ρ.

(c) В выражении из п. (a) этой задачи имеем $w(\pi) = w(l(\pi))\,w(r(\pi))$, поскольку столбец $\binom{j}{k}$ из π либо становится $\binom{j}{k}$ веса w_j/w_k в $l(\pi)$ или $r(\pi)$, либо разлагается на столбцы $\binom{j}{0}$ и $\binom{0}{k}$, имеющие веса z_j/z_0 и z_0/z_k. Если $l(\pi)$ имеет p_j столбцов $\binom{0}{j}$ и q_j столбцов $\binom{j}{0}$, его вес равен $\prod_{j=1}^t (z_j^{q_j} w_j^{n_j-q_j}/z_j^{p_j} w_j^{n_j-p_j}) = \prod_{j=1}^t (w_j/z_j)^{p_j-q_j}$. Теперь $\prod_{j=1}^t (w_j/z_j)^{-q_j}$ есть комплексное сопряжение по отношению к $\prod_{j=1}^t (w_j/z_j)^{q_j}$; таким образом, сумма весов по всем элементам из $P_0(0^k1^{n_1}\dots t^{n_t})$ упрощается:

$$\frac{k!\,(n_1+\cdots+n_t-k)!}{n_1!\dots n_t!}\left|\sum_{p_1+\cdots+p_t=k}\binom{n_1}{p_1}\cdots\binom{n_t}{p_t}\left(\frac{w_1}{z_1}\right)^{p_1}\cdots\left(\frac{w_t}{z_t}\right)^{p_t}\right|^2.$$

Аналогичные соображения применимы и к $r(\pi)$. Полученная сумма положительна, поскольку член для $k = 0$ ненулевой.

23. Можно считать, что исходная цепочка была рассортирована. Пусть $t = 2$, $m = 4$, $w_1 = w_3 = z_1 = z_2 = +1$, $w_2 = w_4 = z_3 = z_4 = -1$ в соотношениях п. (c) из предыдущего упражнения. Тогда $w(\pi) = (-1)^d$, где d — число столбцов $\binom{j}{k}$, для которых $j \ne k$. [См. Gillis, Zeilberger, *European J. Comb.* **4** (1983), 221–223. Впервые этот результат был получен совершенно другим способом; см. Askey, Ismail, Koornwinder, *J. Comb. Theory* **A25** (1978), 277–287. Авторы последней работы отыскали интересные связи между перестановками мультимножеств и интегралами произведений полиномов Лагерра $L_n^\alpha(x) = \sum_{k=0}^n \binom{n+\alpha}{n-k}(-x)^k/k!$.] Аналогичное соотношение для пятибуквенного алфавита несправедливо, поскольку 5! перестановок множества $\{1, 2, 3, 4, 5\}$ включает $1 + 10 + 45$ с четным числом отличий и $0 + 20 + 44$ с нечетным числом.

24. (a) Двойное транспонирование $\begin{smallmatrix}w & x\\ y & z\end{smallmatrix}$ восстанавливает $\begin{smallmatrix}w & x\\ y & z\end{smallmatrix}$. В данном $\operatorname{sort}\binom{x_1\ \dots\ x_n}{y_1\ \dots\ y_n} = \binom{x_1''\ \dots\ x_n''}{y_1''\ \dots\ y_n''}$ порядок сортировки будет нарушен, если найти крайний слева элемент x в верхней строке и транспонировать его влево. В результате будет получен соответствующий y. (Значение $\operatorname{sort}\binom{x_2'\ \dots\ x_n'}{y_2'\ \dots\ y_n'}$ также определяется единственным образом.)

(b) Двухстрочное представление π имеет вид

$$\pi = \operatorname{sort}\begin{pmatrix} y_1 & \dots x_{1n_1} & y_2 & \dots x_{2n_2} & \dots & y_t & \dots x_{tn_t} \\ x_{11} \dots & y_1 & x_{21} \dots & y_2 & \dots & x_{t1} \dots & y_t \end{pmatrix},$$

а результат п. (a) дает нам тот инструмент, который необходим в данном случае. [Если R сохраняет определенные статистики в двухстрочном представлении, то полученная конструкция может быть использована для доказательства средствами комбинаторики некоторых интересных теорем. См. Guo-Niu Han, *Advances in Math.* **105** (1994), 26–41.]

РАЗДЕЛ 5.1.3

1. Достаточно показать, что, подставив это значение в (11), получим, что равенство выполняется при $x = k$, если $k \geq 1$. Используя (7), получим формулу

$$k^n = \sum_{r=0}^{k} \left\langle {n \atop r-1} \right\rangle \binom{k+n-r}{n} = \sum_{0 \leq j \leq r \leq k} (-1)^j (r-j)^n \binom{n+1}{j} \binom{n+k-r}{n}$$
$$= \sum_{s=0}^{k} s^n \sum_{j=0}^{k-s} (-1)^j \binom{n+1}{j} \binom{n+k-s-j}{n}.$$

При $s < k$ суммирование по j можно распространить на диапазон $0 \leq j \leq n+1$, а эта сумма будет равна 0 (это $(n+1)$-я разность полинома n-й степени от j).

2. (a) Число последовательностей $a_1 a_2 \ldots a_n$, которые содержат каждый из элементов $(1, 2, \ldots, q)$ по крайней мере по одному разу, равно $\left\{ {n \atop q} \right\} q!$, как показано в упр. 1.2.6–64; число таких последовательностей, удовлетворяющих условиям, аналогичным (10), при $m = q$ равно $\binom{n-k}{n-q}$, поскольку нужно выбрать $n - q$ из возможных знаков "=". (b) Сложите результаты (a) при $m = n - q$ и $n - q - 1$.

3. Из (20) получаем

$$\sum_n \frac{x^n}{n!} \sum_k \left\langle {n \atop k} \right\rangle (-1)^k = \frac{2}{e^{-2x}+1} = \frac{1}{x}\left(\frac{(-4x)}{e^{-4x}-1} - \frac{(-2x)}{e^{-2x}-1} \right)$$
$$= \frac{1}{x} \sum_{n \geq 0} \frac{B_n x^n}{n!} ((-4)^n - (-2)^n).$$

Следовательно, результат равен $(-1)^{n+1} B_{n+1} 2^{n+1} (2^{n+1} - 1)/(n+1)$. Другой вариант таков: тождество $2/(e^{-2x}+1) = 1 + \tanh x$ позволяет выразить ответ в форме $(-1)^{(n-1)/2} T_n$, если n нечетное, где через T_n обозначены коэффициенты в разложении тангенса по формуле

$$\tan z = T_1 z + T_3 z^3/3! + T_5 z^5/5! + \cdots.$$

Если $n > 0$ четное, то в соответствии со свойством симметрии (7) сумма обращается в 0.

Обратите внимание, что из (18) теперь следует любопытное тождество для чисел Стирлинга:

$$\sum_k \left\{ {n \atop k} \right\} \frac{k!}{(-2)^k} = \frac{2B_{n+1}(1 - 2^{n+1})}{n+1}.$$

4. $(-1)^{n+m} \left\langle {n \atop m} \right\rangle$. (Рассмотрите коэффициент при z^{m+1} в (18).)

5. $\left\langle {p \atop k} \right\rangle \equiv (k+1)^p - k^p \equiv (k+1) - k \equiv 1$ *modulo* p при $0 \leq k < p$. Результат следует из формулы (13), упр. 1.2.6–10 и теоремы 1.2.4F.

6. Нельзя сначала суммировать по k, поскольку слагаемые отличны от 0 при сколь угодно больших j и k, а сумма абсолютных величин бесконечна.

Вот более простой пример, демонстрирующий суть ошибки. Пусть $a_{jk} = (k - j)\,[|j - k| = 1]$. Тогда

$$\sum_{j \geq 0} \left(\sum_{k \geq 0} a_{jk} \right) = \sum_{j \geq 0} (\delta_{j0}) = +1, \quad \text{в то время как} \quad \sum_{k \geq 0} \left(\sum_{j \geq 0} a_{jk} \right) = \sum_{k \geq 0} (-\delta_{k0}) = -1.$$

7. Да. [F. N. David, D. E. Barton, *Combinatorial Chance* (1962), 150–154; см. также ответ к упр. 25.]

8. [*Combinatory Analysis* **1** (1915), 190.] Ответ получается методом включения и исключения. Например, $1/(l_1+l_2)!\,l_3!\,(l_4+l_5+l_6)!$ есть вероятность того, что $x_1<\cdots<x_{l_1+l_2}$, $x_{l_1+l_2+1}<\cdots<x_{l_1+l_2+l_3}$ и $x_{l_1+l_2+l_3+1}<\cdots<x_{l_1+l_2+l_3+l_4+l_5+l_6}$.

Простой $O(n^2)$-алгоритм для подсчета числа перестановок множества $\{1,\ldots,n\}$, которые имеют соответственно длины серий $(l_1,\ldots,l_k)$, был предложен Н. Г. де Брейном (N. G. de Bruijn) в *Nieuw Archief voor Wiskunde* (3) **18** (1970), 61–65.

9. $p_{km}=q_{km}-q_{k(m+1)}$ в (23). Поскольку $\sum_{k,m}q_{km}z^mx^k=\frac{x}{1-x}g(x,z)$ и $g(x,0)=1$, получим

$$h(z,x)=\sum h_k(z)x^k=\frac{x}{1-x}g(x,z)(1-z^{-1})+\frac{x}{1-x}z^{-1}=\frac{(1-z^{-1})x}{e^{(x-1)z}-x}+\frac{z^{-1}x}{1-x}.$$

Таким образом, $h_1(z)=e^z-(e^z-1)/z$; $h_2(z)=(e^{2z}-ze^z)+e^z-(e^{2z}-1)/z$.

10. Пусть $M_n=L_1+\cdots+L_n$ — среднее значение; тогда $\sum M_nx^n=h'(1,x)$, где производная берется по z и равна $x/(e^{x-1}-x)-x/(1-x)=M(x)$. По теореме о вычетах

$$\frac{1}{2\pi i}\oint M(z)z^{-n-1}\,dz=M_n-2(n+\tfrac13)+1+\frac{z_1^{-n}}{z_1-1}+\frac{\bar z_1^{-n}}{\bar z_1-1},$$

если интеграл берется по окружности радиуса r, где $|z_1|<r<|z_2|$. (Обратите внимание на полюс второго порядка в точке $z=1$.) Более того, абсолютное значение этого интеграла меньше, чем $\oint|M(z)|\,r^{-n-1}\,dz=O(r^{-n})$. Интегрируя по окружностям все большего и большего радиуса, получим сходящийся ряд $M_n=2n-\frac13+\sum_{k\ge1}2\Re(1/z_k^n(1-z_k))$.

Для определения дисперсии воспользуемся соотношением $h''(1,x)=-2h'(1,x)-2x(x-1)e^{x-1}/(e^{x-1}-x)^2$. Рассуждая, как при определении среднего значения, но на сей раз по отношению к полюсу третьего порядка, приходим к выводу, что коэффициенты при $h''(1,x)$ асимптотически стремятся к $4n^2+\frac43n-2M_n$ плюс слагаемый большего порядка малости; отсюда получается асимптотическая формула для дисперсии $\frac23n+\frac29$ (плюс экспоненциально уменьшающиеся слагаемые).

11. $P_{kn}=\sum_{t_1\ge1,\ldots,t_k\ge1}D(t_1,\ldots,t_{k-1},n,1)$, где $D(l_1,l_2,\ldots,l_k)$ — определитель Мак-Магона из упр. 8. Разлагая этот определитель по элементам первой строки, получим $P_{kn}=c_0P_{(k-1)n}+c_1P_{(k-2)n}+\cdots+c_{k-2}P_{1n}-E_k(n)$, где c_j и E_k определяются следующим образом:

$$c_j=(-1)^j\sum_{t_1,\ldots,t_{j+1}\ge1}\frac{1}{(t_1+\cdots+t_{j+1})!}=(-1)^j\sum_{m\ge0}\binom{m}{j}\frac{1}{(m+1)!}$$

$$=(-1)^j\sum_{r,m\ge0}\binom{-1}{j-r}\binom{m+1}{r}\frac{1}{(m+1)!}=-1+e\Big(\frac{1}{0!}-\frac{1}{1!}+\cdots+(-1)^j\frac{1}{j!}\Big);$$

$$E_1(n)=1/(n+1)!-1/n!;\qquad E_2(n)=1/(n+1)!;$$

$$E_k(n)=(-1)^k\sum_{m\ge0}\binom{m}{k-3}\frac{1}{(n+2+m)!},\qquad k\ge3.$$

Пусть $P_{0n}=0$, $C(z)=\sum c_jz^j=(e^{1-z}-1)/(1-z)$ и пусть

$$E(z,x)=\sum_{n,k}E_{k+1}(n)z^nx^k=1-e^z+\frac{(e^{1-x}-1)x^2}{(1-x)^2}-\frac{x^2(e^{1-x}-e^z)}{(1-x)(1-x-z)}+\frac{e^z-1-z}{z(1-x)}.$$

Полученное рекуррентное соотношение эквивалентно формуле $C(x)H(z,x)=H(z,x)/x+E(z,x)$; отсюда $H(z,x)=E(z,x)x(1-x)/(xe^{1-x}-1)$. Разлагая в степенной ряд правую часть, придем к соотношению $H_1(z)=h_1(z)$ (см. упр. 9); $H_2(z)=eh_1(z)+1-e^z$.

[*Замечание.* Производящие функции для первых трех серий выведены Кнутом в *CACM* **6** (1963), 685–688. В работе Barton, Mallows, *Ann. Math. Statistics* **36** (1965), 249, выведена формула $1 - H_{n+1}(z) = (1 - H_n(z))/(1 - z) - L_n h_1(z)$ для $n \geq 1$, а также формула (25). Другой подход к решению этой задачи продемонстрирован в упр. 23. Поскольку соседние серии не являются независимыми, не существует простой связи между решенной здесь задачей и более простым результатом упр. 9 (последний, вероятно, и более полезен).]

12. [*Combinatory Analysis* **1** (1915), 209–211.] Число способов, которыми можно разместить элементы мультимножества в t различных ячейках, равно

$$N_t = \binom{t+n_1-1}{n_1}\binom{t+n_2-1}{n_2}\cdots\binom{t+n_m-1}{n_m},$$

поскольку имеется $\binom{t+n_1-1}{n_1}$ способов разместить единицы и т. д. Если потребовать, чтобы ни одна ячейка не пустовала, то, пользуясь методом включения и исключения, найдем, что число способов равно

$$M_t = N_t - \binom{t}{1}N_{t-1} + \binom{t}{2}N_{t-2} - \cdots.$$

Пусть P_k — число перестановок, содержащих k серий; если поместить $k-1$ вертикальных черточек между сериями и $t-k$ дополнительных вертикальных черточек в любые из оставшихся $n-k$ позиций, получим один из M_t способов разделить мультимножество на t непустых различаемых частей. Следовательно,

$$M_t = P_t + \binom{n-t+1}{1}P_{t-1} + \binom{n-t+2}{2}P_{t-2} + \cdots.$$

Приравнивая эти два значения M_t можно выразить $P_1, P_2, \ldots$ последовательно в терминах $N_1, N_2, \ldots$. (Желательно отыскать более прямое доказательство.)

13. $1 + \frac{1}{2}13 \times 3 = 20.5$.

14. Как следует из найденного Фоатой соотношения, данная перестановка соответствует

$$(3\,1) \top (1) \top \cdots \top (4) = \begin{pmatrix} 1&1&1&1&2&2&2&2&3&3&3&3&4&4&4&4 \\ 3&1&1&2&3&4&3&2&1&1&3&4&2&2&4&4 \end{pmatrix};$$

но согласно (33) она соответствует

$$\begin{pmatrix} 1&1&1&1&2&2&2&2&3&3&3&3&4&4&4&4 \\ 2&4&4&3&3&3&1&1&4&4&2&1&2&1&2&3 \end{pmatrix},$$

которая, в свою очередь, соответствует 2 3 4 2 3 4 1 4 2 1 4 3 2 1 3 1. Последняя же имеет 9 серий.

15. Число перемежающихся серий равно увеличенному на единицу числу индексов j, таких, что $1 < j < n$ и либо $a_{j-1} < a_j > a_{j+1}$, либо $a_{j-1} > a_j < a_{j+1}$. При фиксированном j вероятность равна $\frac{2}{3}$; следовательно, среднее значение при $n \geq 2$ равно $1 + \frac{2}{3}(n-2)$.

16. Каждая перестановка множества $\{1, 2, \ldots, n-1\}$, которая имеет k перемежающихся серий, при вставке нового элемента n во все возможные позиции порождает k перестановок с k такими сериями, 2 перестановки с $k+1$ сериями и $n-k-2$ перестановки с $k+2$ сериями. Следовательно,

$$\left|{n \atop k}\right| = k\left|{n-1 \atop k}\right| + 2\left|{n-1 \atop k-1}\right| + (n-k)\left|{n-1 \atop k-2}\right|.$$

Удобно положить $\left|{1 \atop k}\right| = \delta_{k0}$, $G_1(z) = 1$. Тогда

$$G_n(z) = \frac{z}{n}\big((1-z^2)G'_{n-1}(z) + (2 + (n-2)z)G_{n-1}(z)\big).$$

Дифференцирование приводит нас к рекуррентному соотношению для $x_n = G_n'(1)$:

$$x_n = \frac{1}{n}((n-2)x_{n-1} + 2n - 2).$$

Его решение при $n \geq 2$ имеет вид $x_n = \frac{2}{3}n - \frac{1}{3}$. Еще одно дифференцирование приведет к рекуррентному соотношению для $y_n = G_n''(1)$:

$$y_n = \frac{1}{n}((n-4)y_{n-1} + \tfrac{8}{3}n^2 - \tfrac{26}{3}n + 6).$$

Положим $y_n = \alpha n^2 + \beta n + \gamma$ и, решая уравнения относительно α, β, γ, получим $y_n = \frac{4}{9}n^2 - \frac{14}{15}n + \frac{11}{90}$ при $n \geq 4$. Следовательно, $\mathrm{var}(g_n) = \frac{1}{90}(16n - 29)$, $n \geq 4$.

Эти формулы для математического ожидания и дисперсии получены Ж. Бьенэйме (J. Bienaymé) и приведены без доказательства [*Bull. Soc. Math. de France* **2** (1874), 153–154; *Comptes Rendus Acad. Sci. Paris* **81** (1875), 417–423, см. также замечание Бертрана (Bertrand) на с. 458]. Рекуррентное соотношение для $\langle\langle {n \atop k} \rangle\rangle$ получено Д. Андрэ (D. André) [*Comptes Rendus Acad. Sci. Paris* **97** (1883), 1356–1358; *Annales Scientifiques de l'École Normale Supérieure* (3) **1** (Paris, 1884), 121–134]. Андрэ обратил внимание на то, что $g_n(-1) = 0$ при $n \geq 4$. Таким образом, число перестановок с четным числом перемежающихся серий равно $n!/2$. Он также доказал формулу для математического ожидания и определил количество перестановок, которые имеют максимальное число перемежающихся серий (см. упр. 5.1.4–23). Можно также показать, что

$$G_n(z) = \left(\frac{1+z}{2}\right)^{n-1}(1+w)^{n+1} g_n\left(\frac{1-w}{1+w}\right), \qquad w = \sqrt{\frac{1-z}{1+z}}, \qquad n \geq 2,$$

где $g_n(z)$ — производящая функция (18) для восходящих серий. [См. F. N. David, D. E. Barton, *Combinatorial Chance* (London: Griffin, 1962), 157–162.]

17. ${n+1 \atop 2k-1}$; ${n \atop 2k-2}$ последовательностей, заканчивающихся элементом 0, а ${n \atop 2k-1}$ последовательностей, заканчивающихся элементом 1.

18. (a) Пусть данная последовательность — таблица инверсий, которая была рассмотрена в разделе 5.1.1. Если в этой последовательности имеется k нисходящих серий, инверсия соответствующей перестановки имеет k нисходящих серий (см. ответ к упр. 5.1.1–8(e)); следовательно, ответ для данной задачи — $\left\langle {n \atop k} \right\rangle$. (b) Это число удовлетворяет равенству $f(n,k) = kf(n-1,k) + (n-k+1)f(n-1,k-1)$, значит, оно должно быть равно $\left\langle {n \atop k-1} \right\rangle$. [См. D. Dumont, *Duke Math. J.* **41** (1974), 313–315.]

19. (a) $\left\langle {n \atop k} \right\rangle$ в соответствии с теоремой 5.1.2B. (b) Существует $(n-k)!$ способов расположения $n-k$ последующих неатакующих ладей на всей доске; следовательно, ответ — $1/(n-k)!$ раз $\sum_{j\geq 0} a_{nj}\binom{j}{k}$, где $a_{nj} = \left\langle {n \atop j} \right\rangle$ — решения для случая (a). Это приводит к $\left\{ {n \atop n-k} \right\}$, принимая во внимание результат упр. 2. [В гл. 5 книги Риордана *Introduction to Combinatorial Analysis* (Wiley, 1958) анализируется общий случай размещения ладей.]

В прямом доказательстве этого результата, авторство которого принадлежит Э. А. Бендеру (E. A. Bender), каждое разбиение множества $\{1, 2, \ldots, n\}$ на k непустых непересекающихся подмножеств соответствует некоторому расположению $n-k$ ладей. Пусть разбиение таково:

$$\{1, 2, \ldots, n\} = \{a_{11}, a_{12}, \ldots, a_{1n_1}\} \cup \cdots \cup \{a_{k1}, \ldots, a_{kn_k}\},$$

где $a_{ij} < a_{i(j+1)}$ при $1 \leq j < n_i$, $1 \leq i \leq k$. Ему соответствует следующее расположение ладей: ладьи помещаются в a_{ij}-й столбец $a_{i(j+1)}$-й строки, где $1 \leq j \leq n_i$, $1 \leq i \leq k$. Например, позиция, приведенная на рис. 4, соответствует разбиению $\{1, 3, 8\} \cup \{2\} \cup \{4, 6\} \cup \{5\} \cup \{7\}$.

20. Число чтений равно числу серий в обратной перестановке. Первая серия соответствует первому чтению и т. д.

21. Перестановка имеет $n+1-k$ серий и требует $n+1-j$ чтений.

22. [*J. Combinatorial Theory* **1** (1966), 350–374.] Если $rs < n$, то при некотором чтении выбирается $t > r$ элементов, $a_{i_1} = j+1, \ldots, a_{i_t} = j+t$, где $i_1 < \cdots < i_t$. Нельзя получить $a_m > a_{m+1}$ для всех m в диапазоне $i_k \le m < i_{k+1}$. Таким образом, перестановка содержит, по меньшей мере, $t-1$ таких мест, где $a_m < a_{m+1}$; отсюда следует, что в перестановке не более $n-t+1$ серий.

С другой стороны, рассмотрим перестановку $\alpha_r \ldots \alpha_2\, \alpha_1$, в которой блок α_j содержит числа $\equiv j$ (по модулю r) в порядке убывания; например, когда $n=9$ и $r=4$, эта перестановка имеет вид 8 4 7 3 6 2 9 5 1. Если $n \ge 2r-1$, эта перестановка имеет $r-1$ возрастаний. Значит, в ней имеется $n+1-r$ серий. Более того, если $r > 1$, она требует точно $n+1-\lceil n/r\rceil$ чтений. Можно по-другому расставить элементы в $\{kr+1, \ldots, kr+r\}$, не меняя числа серий; таким способом можно уменьшить число чтений до любой желаемой величины $\ge \lceil n/r\rceil$.

Теперь предположим, что $rs \ge n$, $r+s \le n+1$ и $r, s \ge 1$. Как показано в упр. 20 и 21, можно считать, что $r \le s$, поскольку отражение инверсии с $n+1-r$ сериями и s чтениями имеет $n+1-s$ серий и r чтений. После этого рассуждения предыдущего абзаца можно применить ко всем случаям, кроме тех, где $s > n+1-\lceil n/r\rceil$ и $r \ge 2$. Чтобы завершить доказательство, можно воспользоваться перестановкой в форме

$$2k{+}1\ 2k{-}1\ \ldots\ 1\ n{+}2{-}r\ n{+}1{-}r\ \ldots\ 2k{+}2\ 2k\ \ldots\ 2\ n{+}3{-}r\ \ldots\ n{-}1\ n,$$

которая имеет $n+1-r$ серий и $n+1-r-k$ чтений, $0 \le k \le \frac{1}{2}(n-r)$.

23. [*SIAM Review* **3** (1967), 121–122.] Допустим, что бесконечная перестановка состоит из независимых частей, подчиненных равномерному закону распределения. Пусть $f_k(x)\,dx$ — вероятность того, что k-я удлиненная серия начинается с x, и пусть $g(u,x)\,dx$ — вероятность того, что удлиненная серия начинается с x, при условии, что предыдущая удлиненная серия начинается с u. Тогда $f_1(x) = 1$, $f_{k+1}(x) = \int_0^1 f_k(u)g(u,x)\,du$. Здесь $g(u,x) = \sum_{m\ge1} g_m(u,x)$, где

$$\begin{aligned} g_m(u,x) &= \Pr(u < X_1 < \cdots < X_m > x \ \text{ или } \ u > X_1 > \cdots > X_m < x) \\ &= \Pr(u < X_1 < \cdots < X_m) + \Pr(u > X_1 > \cdots > X_m) \\ &\qquad - \Pr(u < X_1 < \cdots < X_m < x) - \Pr(u > X_1 > \cdots > X_m > x) \\ &= (u^m + (1-u)^m + |u-x|^m)/m!\,. \end{aligned}$$

Следовательно, $g(u,x) = e^u + e^{1-u} - 1 - e^{|u-x|}$ и получим $f_2(x) = 2e - 1 - e^x - e^{1-x}$. Можно показать, что $f_k(x)$ стремится к предельному значению $(2\cos(x-\frac{1}{2}) - \sin\frac{1}{2} - \cos\frac{1}{2})/(3\sin\frac{1}{2} - \cos\frac{1}{2})$. Средняя длина серии, начинающейся с x, равна $e^x + e^{1-x} - 1$; таким образом, длина k-ой удаленной серии $\mathcal{L}_k$ равна $\int_0^1 f_k(x)(e^x + e^{1-x} - 1)\,dx$; $\mathcal{L}_1 = 2e - 3 \approx 2.43656$; $\mathcal{L}_2 = 3e^2 - 8e + 2 \approx 2.42091$. Аналогичные результаты имеются в разделе 5.4.1.

24. Рассуждая, как и раньше, получим в результате

$$1 + \sum_{0\le k<n} 2^k (p^2+q^2)^k \big(p^2 + 2pq(2^{n-k-1} - 1 + q^2((2pq)^{n-k-1} - 1)/(2pq-1))\big).$$

После суммирования и упрощения получим искомую функцию

$$2^n(p^2+q^2)^n\big(p(p-q)/(p^2+q^2-pq) - \tfrac{1}{2}\big) + (2pq)^n pq^3/(p^2+q^2)(p^2+q^2-pq) \\ + q^2/(p^2+q^2) + 2^{n-1}.$$

25. Пусть $V_j = (U_1 + \cdots + U_j) \bmod 1$; тогда $V_1, \ldots, V_n$ — независимые равномерно распределенные числа в интервале $[0\,..\,1)$, образующие перестановку, которая имеет k нисходящих

серий тогда и только тогда, когда $\lfloor U_1 + \cdots + U_n \rfloor = k$. Отсюда получается ответ $\left\langle {n \atop k} \right\rangle / n!$; это свойство впервые обнаружил С. Тэнни (S. Tanny) [см. *Duke Math. J.* **40** (1973), 717–722].

26. Например, $\vartheta^5(1-z)^{-1} = (z + 26z^2 + 66z^3 + 26z^4 + z^5)/(1-z)^6$.

27. Следующее правило дает взаимно однозначное соответствие между перестановкой $a_1\, a_2 \ldots a_n$, имеющей k нисходящих серий, и n-узловым разрастающимся лесом, имеющим $k+1$ лист. Первый корень — a_1, а его потомки — лес, соответствующий $a_2 \ldots a_k$, где k — наименьшее число, удовлетворяющее $a_{k+1} < a_1$, или $k = n$. [R. P. Stanley, *Enumerative Combinatorics* **1** (Wadsworth, 1986), Proposition 1.3.16.]

РАЗДЕЛ 5.1.4

1.

1	2	3	8
4	5	7	
6	9		

1	3	5	8
2	4	9	
6	7		

; $\begin{pmatrix} 1 & 3 & 4 & 5 & 7 & 8 & 9 \\ 5 & 9 & 2 & 4 & 8 & 1 & 7 \end{pmatrix}$.

2. Пусть p_j — элемент столбца $t-1$, если p_i вставляется в столбец t. Тогда (q_j, p_j) принадлежит классу $t-1$, $q_j < q_i$, $p_j < p_i$; таким образом, по индукции приходим к заключению, что существуют индексы $i_1, \ldots, i_t$, обладающие нужным свойством. Обратно, если $q_j < q_i$, $p_j < p_i$ и (q_j, p_j) принадлежит классу $t-1$, то, когда вставляется p_i, столбец $t-1$ содержит элемент $< p_i$. Таким образом, (q_i, p_i) принадлежит классу $\geq t$.

3. Здесь столбцы представляют собой последовательные "вытеснения" в смысле (9), которые образуются при вставке p_i. Строки 1 и 2 отражают операции над строкой 1 (ср. с (14)). Если убрать столбцы, во второй строке которых стоит элемент ∞, то строки 0 и 2 образуют "вытесненный" массив, как в (15). Сформулированный метод перехода от строки k к строке $k+1$ — это не что иное, как описанный в данном разделе алгоритм определения классов.

4. (a) Проанализируйте разные случаи. Рассмотрите сначала воздействие на строку 1 и воздействие на последовательность элементов, вытесненных из строки 1, а затем распространите анализ по индукции на диаграмму заданного размера. (b) При помощи допустимых обменов можно смоделировать операции алгоритма I, представив диаграмму до и после выполнения процедуры в виде канонической перестановки. Например, рядом допустимых обменов можно трансформировать

$$17\ 11\ 4\ 13\ 14\ 2\ 6\ 10\ 15\ 1\ 3\ 5\ 9\ 12\ 16\ 8$$

в

$$17\ 11\ 13\ 4\ 10\ 14\ 2\ 6\ 9\ 15\ 1\ 3\ 5\ 8\ 12\ 16$$

(см. (4) и (5)).

5. Допустимые обмены симметричны в направлении слева направо, а каноническая перестановка для P очевидным образом переходит в P^T, если вставлять элементы в обратном порядке.

6. Будем считать, что существует всего t классов; из них точно k имеют нечетное число элементов, поскольку элементы класса имеют вид

$$(p_{i_k}, p_{i_1}), \qquad (p_{i_{k-1}}, p_{i_2}), \qquad \ldots, \qquad (p_{i_1}, p_{i_k})$$

(см. (18) и (22)). Вытесненный двухстрочный массив имеет ровно $t-k$ фиксированных точек, что следует из способа его формирования. Следовательно, по индукции диаграмма без ее первой строки имеет $t-k$ столбцов нечетной длины. Таким образом, t элементов в первой строке приводят к появлению k столбцов нечетной длины по всем диаграммам.

7. Число столбцов, а именно — длина строки 1, равно числу классов (упр. 2). Число строк равно числу столбцов в P^T. Далее, применив результаты упр. 5 (или теоремы D), завершаем доказательство.

8. Диаграмма P, в которой содержится более n^2 элементов, должна иметь либо более n строк, либо более n столбцов. Однако существует и диаграмма размером $n \times n$. [Этот результат впервые был доказан в *Compositio Math.* **2** (1935), 463–470.]

9. Подобные перестановки обладают взаимно однозначным соответствием с парами диаграмм вида $(n, n, \ldots, n)$; таким образом, с учетом (34) ответ таков:

$$\left(\frac{n^2!\,\Delta(2n-1, 2n-2, \ldots, n)}{(2n-1)!\,(2n-2)!\ldots n!}\right)^2 = \left(\frac{n^2!}{(2n-1)(2n-2)^2\ldots n^n(n-1)^{n-1}\ldots 1^1}\right)^2.$$

Воистину удивительно, что решением этой задачи является такая простая формула. Теперь можно подсчитать число перестановок $\{1, 2, \ldots, mn\}$, в которых отсутствуют возрастающие последовательности длиннее m и убывающие последовательности длиннее n.

10. Доказываем по индукции, что на шаге S3 оба массива — и $P_{(r-1)s}$, и $P_{r(s-1)}$ — меньше, чем $P_{(r+1)s}$ и $P_{r(s+1)}$.

11. Разумеется, нам еще нужно знать, какой элемент был раньше на месте P_{11}. Тогда существует возможность восстановить исходный вид диаграммы, используя алгоритм, чрезвычайно похожий на алгоритм S.

12. $\binom{n_1+1}{2} + \binom{n_2+2}{2} + \cdots + \binom{n_m+m}{2} - \binom{m+1}{3}$ — это путь, пройденный в общей сложности. Минимальное значение равно сумме первых n членов последовательности $1, 2, 2, 3, 3, 3, 4, 4, 4, 4\ldots$ в упр. 1.2.4–41; эта сумма равна приблизительно $\sqrt{8/9}\,n^{3/2}$. (Почти для всех диаграмм из n элементов оценка значения минимума подходит очень близко к этой границе, как следует из упр. 29. Поэтому среднее число применений шага S3 равно $\Theta(n^{3/2})$.)

13. Пусть в перестановке участвуют элементы множества $\{1, 2, \ldots, n\}$, так что $a_i = 1$, и предположим, что $a_j = 2$. *Случай 1*, $j < i$. Тогда 1 вытесняет 2, а значит, строка 1 диаграммы, соответствующей перестановке $a_1 \ldots a_{i-1}\, a_{i+1} \ldots a_n$, есть строка 1 в P^S; вытесненная перестановка — та же, что и прежняя вытесненная перестановка, но ее наименьший элемент теперь равен 2; результат по индукции можно расширить и на n. *Случай 2*, $j > i$. Применим результат случая 1 к P^T, приняв во внимание результат упр. 5 и тот факт, что $(P^T)^S = (P^S)^T$.

15. Как и в (37), приведенная в качестве примера перестановка соответствует диаграмме

1	2	5	9	11
3	6	7		
4	8	10		

;

следовательно, искомое число равно $f(l, m, n) = (l+m+n)!\,(l-m+1)(l-n+2)(m-n+1)/(l+2)!\,(m+1)!\,(n)!$ при условии, разумеется, что $l \geq m \geq n$.

16. Как следует из теоремы H, 80 080 способами.

17. Поскольку полином g антисимметричен по x, он обращается в 0 при $x_i = x_j$; следовательно, он делится на $x_i - x_j$ при всех $i < j$. Таким образом, $g(x_1, \ldots, x_n; y) = h(x_1, \ldots, x_n; y)\Delta(x_1, \ldots, x_n)$. Здесь функция h должна быть однородным полиномом первой степени $x_1, \ldots, x_n, y$, симметричным относительно $x_1, \ldots, x_n$, так что $h(x_1, \ldots, x_n; y) = a(x_1 + \cdots + x_n) + by$ для некоторых a и b, зависящих только от n. Можно вычислить a,

положив $y = 0$; можно вычислить b, взяв частную производную по y и затем положив $y = 0$. Получим

$$\frac{\partial}{\partial y}\Delta(x_1,\ldots,x_i+y,\ldots,x_n)|_{y=0} = \frac{\partial}{\partial x_i}\,\Delta(x_1,\ldots,x_n) = \Delta(x_1,\ldots,x_n)\sum_{j\neq i}\frac{1}{x_i-x_j}.$$

Окончательный результат таков:

$$\sum_i\sum_{j\neq i}(x_i/(x_i-x_j)) = \sum_i\sum_{j<i}(x_i/(x_i-x_j)+x_j/(x_j-x_i)) = \binom{n}{2}.$$

18. Она должна равняться $\Delta(x_1,\ldots,x_n)\cdot(b_0+b_1y+\cdots+b_my^m)$, где все b_k — однородные симметричные полиномы степени $m-k$ от переменных x. Имеем

$$\frac{\partial^k}{k!\,\partial y^k}\,\Delta(x_1,\ldots,x_i+y,\ldots,x_n)|_{y=0} = \Delta(x_1,\ldots,x_n)\sum\Big(1\Big/\prod_{l=1}^k(x_i-x_{j_l})\Big),$$

где сумма берется по всем $\binom{n-1}{k}$ способам выбора различных индексов $j_1,\ldots,j_k\neq i$. Теперь в выражении $b_k = \sum x_i^m/\prod_{l=1}^k(x_i-x_{j_l})$ можно скомбинировать группы из $k+1$ членов, имеющих данный набор индексов $\{i,j_1,\ldots,j_k\}$; например, при $k=2$ сгруппируем наборы из трех членов вида $a^m/(a-b)(a-c)+b^m/(b-a)(b-c)+c^m/(c-a)(c-b)$. Сумма членов каждой группы вычисляется, как в упр. 1.2.3–33: $[z^{m-k}]\,1/(1-x_iz)(1-x_{j_1}z)\ldots(1-x_{j_k}z)$. Таким образом, находим, что

$$b_k = \sum_j\binom{n-j}{k+1-j}\sum s(p_1,\ldots,p_j),$$

где $s(p_1,\ldots,p_j)$ — симметричная функция, содержащая всевозможные различные одночлены вида $x_{i_1}^{p_1}\ldots x_{i_j}^{p_j}$, с различными индексами $i_1,\ldots,i_j\in\{1,\ldots,n\}$, а внутренняя сумма берется по всем разбиениям $m-k$ точно на j частей, таких, что $p_1\geq\cdots\geq p_j\geq 1$, $p_1+\cdots+p_j=m-k$. (Этот результат получен совместно с Э. А. Бендером в 1969 году.)

При $m=2$ ответ равен $\big(s(2)+(n-1)s(1)y+\binom{n}{3}y^2\big)\,\Delta(x_1,\ldots,x_n)$; при $m=3$ получаем $\big(s(3)+((n-1)s(2)+s(1,1))y+\binom{n-1}{2}s(1)y^2+\binom{n}{4}y^3\big)\,\Delta(x_1,\ldots,x_n)$ и т. д.

В другом выражении b_k представляет собой коэффициент при z^m в выражении

$$\left(\binom{n}{k+1}z^k-\binom{n-1}{k+1}a_1z^{k+1}+\binom{n-2}{k+1}a_2z^{k+2}-\cdots\right)\Big/(1-a_1z+a_2z^2-\cdots),$$

где $a_l=\sum_{1\leq i_1<\cdots<i_l\leq n}x_{i_1}\ldots x_{i_l}$ — элементарная симметричная функция. Умножение на y^k и суммирование по k дает ответ в виде коэффициента при z^m в выражении

$$\frac{1}{yz}\left(\frac{(1+z(y-x_1))\ldots(1+z(y-x_n))}{(1-zx_1)\ldots(1-zx_n)}-1\right)\Delta(x_1,\ldots,x_n).$$

19. Пусть транспонированная диаграмма имеет форму $(n_1',n_2',\ldots,n_r')$; ответ будет таким:

$$\frac{1}{2}f(n_1,n_2,\ldots,n_m)\left(\frac{(\sum n_i^2-\sum n_j'^2)}{n(n-1)}+1\right),$$

где $n=\sum n_i=\sum n_j'$. (Эту же формулу можно выразить в менее симметричной форме, воспользовавшись соотношением $\sum in_i=\frac{1}{2}(n+\sum n_j'^2)$.)

Замечание. В работе W. Feit, *Proc. Amer. Math. Soc.* **4** (1953), 740–744, показано, что число способов размещения целых чисел $\{1,2,\ldots,n\}$ в массиве, который является "разностью" двух форм диаграмм $(n_1,\ldots,n_m)\backslash(l_1,\ldots,l_m)$, где $0\leq l_j\leq n_j$ и $n=\sum(n_j-l_j)$, равно $n!\det(1/((n_j-j)-(l_i-i))!)$.

20. Аргумент, который был отвергнут при обсуждении теоремы H, в данном случае справедлив — соответствующие события *действительно* независимы.

Замечание. Если рассмотреть все $n!$ способов маркировки узлов, то рассмотренные здесь маркировки представляют собой маркировки, не имеющие "инверсий". Инверсии в перестановках — это то же самое, что инверсии маркировок дерева в особых случаях, когда дерево является просто путем. [См. A. Björner, M. L. Wachs, *J. Combinatorial Theory* **A52** (1989), 165–187.]

21. [*Michigan Math. J.* **1** (1952), 81–88.] Пусть $g(n_1,\ldots,n_m) = (n_1+\cdots+n_m)!\ \Delta(n_1,\ldots,n_m)/n_1!\ldots n_m!\,\sigma(n_1,\ldots,n_m)$, где $\sigma(x_1,\ldots,x_m) = \prod_{1\le i<j\le m}(x_i+x_j)$. Для того чтобы доказать, что $g(n_1,\ldots,n_m)$ равно числу способов заполнения сдвинутой диаграммы, нужно сначала доказать, что $g(n_1,\ldots,n_m) = g(n_1-1,\ldots,n_m)+\cdots+g(n_1,\ldots,n_m-1)$. Тождество, соответствующее упр. 17, имеет вид $x_1\Delta(x_1+y,\ldots,x_n)/\sigma(x_1+y,\ldots,x_n)+\cdots+x_n\Delta(x_1,\ldots,x_n+y)/\sigma(x_1,\ldots,x_n+y) = (x_1+\cdots+x_n)\Delta(x_1,\ldots,x_n)/\sigma(x_1,\ldots,x_n)$. Оно не зависит от y; если вычислить производную, как в упр. 17, можно обнаружить, что $2x_ix_j/(x_j^2-x_i^2)+2x_jx_i/(x_i^2-x_j^2) = 0$.

22. Будем считать, что $m = N$, добавив к исходной форме нули, если в этом возникнет необходимость; если $m > N$ и $n_m > 0$, число способов, очевидно, равно 0. При $m = N$ ответ таков:

$$\det\begin{pmatrix} \binom{n_1+m-1}{m-1} & \binom{n_2+m-2}{m-1} & \cdots & \binom{n_m}{m-1} \\ \vdots & \vdots & & \vdots \\ \binom{n_1+m-1}{0} & \binom{n_2+m-2}{0} & \cdots & \binom{n_m}{0} \end{pmatrix}.$$

Доказательство. Можно считать, что $n_m = 0$, так что, если $n_m > 0$, первые n_m столбцов массива должны быть заполнены числами i в строке i, и тогда можно рассмотреть оставшуюся форму $(n_1-n_m,\ldots,n_m-n_m)$. Применив индукцию по m, получим число способов

$$\sum_{\substack{n_2\le k_1\le n_1\\ \vdots \\ n_m\le k_{m-1}\le n_{m-1}}} \det\begin{pmatrix} \binom{k_1+m-2}{m-2} & \binom{k_2+m-3}{m-2} & \cdots & \binom{k_{m-1}}{m-2} \\ \vdots & \vdots & & \vdots \\ \binom{k_1+m-2}{0} & \binom{k_2+m-3}{0} & \cdots & \binom{k_{m-1}}{0} \end{pmatrix},$$

где n_j-k_j представляет число элементов m в строке j. Суммирование по каждому индексу k_j можно выполнить независимо; в итоге получим

$$\det\begin{pmatrix} \binom{n_1+m-1}{m-1}-\binom{n_2+m-2}{m-1} & \binom{n_2+m-2}{m-1}-\binom{n_3+m-3}{m-1} & \cdots & \binom{n_{m-1}+1}{m-1}-\binom{n_m}{m-1} \\ \vdots & \vdots & & \vdots \\ \binom{n_1+m-1}{1}-\binom{n_2+m-2}{1} & \binom{n_2+m-2}{1}-\binom{n_3+m-3}{1} & \cdots & \binom{n_{m-1}+1}{1}-\binom{n_m}{1} \end{pmatrix}.$$

Это и есть искомый ответ, поскольку $n_m = 0$. Полученный результат можно привести к определителю Вандермонда при помощи операции над строками: $\Delta(n_1+m-1, n_2+m-2, \ldots, n_m)/(m-1)!\,(m-2)!\ldots 0!$. [Ответ к этому упражнению, полученный при решении аналогичной задачи из теории групп, приводится в работе D. E. Littlewood *Theory of Group Characters* (Oxford, 1940), 189.]

23. [*Journal de Math.* (3) **7** (1881), 167–184.] (Это частный случай упр. 5.1.3–8, когда все серии имеют длину 2, кроме, вероятно, последней, которая может иметь длину 1.) Если $n \ge 2$, элемент n должен оказаться на одной из крайних справа позиций в какой-либо строке. Если поместить его в крайнюю справа клетку стоки k, можно получить

$\binom{n-1}{2k-1}A_{2k-1}A_{n-2k}$ способов заполнения остальных клеток. Пусть

$$h(z) = \sum_{n\geq 1} A_{2n-1}z^{2n-1}/(2n-1)! = \tfrac{1}{2}(g(z) - g(-z));$$

тогда

$$h(z)g(z) = \sum_{k,n\geq 1}\binom{n}{2k-1}A_{2k-1}A_{n-2k+1}z^n/n! = \left(\sum_{n\geq 1}A_{n+1}z^n/n!\right) - 1 = g'(z) - 1.$$

В выражении для $h(z)y(z)$ заменим z на $-z$, сложим исходное и преобразованное выражения — и получится $h(z)^2 = h'(z) - 1$; следовательно, $h(z) = \tan z$. Полагая $k(z) = g(z) - h(z)$, имеем $h(z)\,k(z) = k'(z)$; значит, $k(z) = \sec z$ и $g(z) = \sec z + \tan z = \tan(\frac{1}{2}z + \frac{1}{4}\pi)$. Коэффициенты A_{2n}, таким образом, — это числа Эйлера $|E_{2n}|$, а коэффициенты A_{2n-1} — это числа тангенса $T_{2n-1} = (-1)^{n-1}4^n(4^n-1)B_{2n}/(2n)$ (см. упр. 5.1.3–3). Таблицы этих чисел имеются в *Math. Comp.* **21** (1967), 663–688; начало последовательности $(A_0, A_1, A_2, \ldots) = (1, 1, 1, 2, 5, 16, 61, 272, 1385, 7936, \ldots)$. Самый простой способ вычисления числа тангенса и чисел Эйлера — это, возможно, построение треугольного массива

$$\begin{array}{ccccccc}
 & & & 1 & & & \\
 & & 0 & & 1 & & \\
 & 1 & & 1 & & 0 & \\
0 & & 1 & & 2 & & 2 \\
5 & & 5 & & 4 & & 2 \\
0 & & 5 & & 10 & & 14 \\
61 & & 61 & & 56 & & 46 \\
 & 32 & & 16 & & 0 & \\
 & & 16 & & 0 & &
\end{array},$$

в котором частичные суммы попеременно формируются слева направо и справа налево [A. J. Kempner, *Tôhoku Math. J.* **37** (1933), 348–349].

25. В общем случае, если u_{nk} — число перестановок множества $\{1, 2, \ldots, n\}$, не содержащих циклов, длина которых больше k, то $\sum u_{nk}z^n/n! = \exp(z + z^2/2 + \cdots + z^k/k)$; это можно доказать, перемножив $\exp(z) \times \cdots \times \exp(z^k/k)$, в результате чего получится

$$\sum_n z^n\left(\sum_{j_1+2j_2+\cdots+kj_k=n} 1/1^{j_1}j_1!\,2^{j_2}j_2!\ldots\right)$$

(см. также упр. 1.3.3–21). Аналогично $\exp(\sum_{s\in S} z^s/s)$ — соответствующая производящая функция для перестановок, длины циклов которых принадлежат данному множеству S.

26. Интеграл от 0 до ∞ равен $n^{(t+1)/4}\Gamma((t+1)/2)/2^{(t+3)/2}$, потому что его можно свести к гамма-функции (в упр. 1.2.5–20 $t = 2x^2/\sqrt{n}$). Таким образом, интегрируя в пределах от $-\infty$ до ∞, получим 0, если t нечетное, в противном случае — $n^{(t+1)/4}\sqrt{\pi}\,t!/2^{(3t+1)/2}(t/2)!$.

27. (a) Если $r_i < r_{i+1}$ и $c_i < c_{i+1}$, то неравенства $i < Q_{r_ic_{i+1}} < i+1$ несправедливы. Если $r_i \geq r_{i+1}$ и $c_i \geq c_{i+1}$, мы, определенно, не получим $i+1 \leq Q_{r_ic_{i+1}} \leq i$. (b) Докажите по индукции по числу строк в диаграмме для $a_1 \ldots a_i$, что из неравенства $a_i < a_{i+1}$ следует $c_i < c_{i+1}$, а из $a_i > a_{i+1}$ следует $c_i \geq c_{i+1}$. (Рассмотрите строку 1 и последовательность "вытесненных" элементов.) (c) Это следует из теоремы D, (c).

28. Данный результат получен А. М. Вершиком и С. В. Керовом (*Докл. АН СССР* **233** (1977), 1024–1028); см. также B. F. Logan, L. A. Shepp, *Advances in Math.* **26** (1977), 206–222. [Дж. Байк (J. Baik), П. Дейфт (P. Deift) и К. Йохансон (K. Johansson) показали в 1998 году, что среднеквадратичное отклонение равно $\Theta(n^{1/6})$; более того, вероятность того, что длина меньше $2\sqrt{n} + tn^{1/6}$, стремится к $\exp(-\int_t^\infty (x-t)u^2(x)\,dx)$, где $u''(x) = 2u^3(x) + xu(x)$ и $u(x)$ — асимптоты функций Айри $\mathrm{Ai}(x)$ as $x \to \infty$.]

29. Среднее число возрастающих подпоследовательностей длиной l равно $\binom{n}{l}/l!$. (Как следует из упр. 8 и 29, вероятность того, что самая длинная возрастающая последовательность имеет длину $\geq e\sqrt{n}$ или $\leq \sqrt{n}/e$, равна $O(1/\sqrt{n})$. [J. D. Dixon, *Discrete Math.* **12** (1975), 139–142.]

30. [*Discrete Math.* **2** (1972), 73–94; упрощенное доказательство принадлежит Марку ван Льювену (см. Marc van Leeuwen, *Electronic J. Combinatorics* **3**, 2 (1996), paper #R15).]

31. $x_n = a_{\lfloor n/2 \rfloor}$, где $a_0 = 1$, $a_1 = 2$, $a_n = 2a_{n-1} + (2n-2)a_{n-2}$; $\sum a_n z^n/n! = \exp(2z + z^2) = (\sum t_n z^n/n!)^2$; $x_n \approx \exp(\frac{1}{4}n \ln n - \frac{1}{4}n + \sqrt{n} - \frac{1}{2} - \frac{1}{2}\ln 2)$ при четном n. [См. E. Lucas, *Théorie des Nombres* (1891), 217–223.]

32. Пусть $m_n = \int_{-\infty}^{\infty} t^n e^{-(t-1)^2/2} dt/\sqrt{2\pi}$. Тогда $m_0 = m_1 = 1$ и $m_{n+1} - m_n = nm_{n-1}$, если интегрировать по частям. Таким образом, $m_n = t_n$ вследствие (40).

33. Верно; это равно $\det_{i,j=1}^{m} \binom{a_i}{j-1}$. [Митчелл (Mitchell) в *Amer. J. Math.* **4** (1881), 341–344, показал, что это — число членов разложения симметричной функции, которая теперь называется функцией Шура. Конечно, если $0 < a_1 < \cdots < a_m$, это будет число членов в $S_{n_1 n_2 \ldots n_m}(x_1, x_2, \ldots, x_m)$, где $n_1 = a_m - m$, $n_2 = a_{m-1} - (m-1)$, $\ldots$, $n_m = a_1 - 1$. Данная функция Шура представляет собой сумму по всем обобщенным диаграммам формы $(n_1, \ldots, n_m)$ с элементами в $\{1, \ldots, m\}$ произведений x_j для всех j в диаграмме. Здесь обобщенная диаграмма — это то же самое, что обычная диаграмма, но в строке допустимо присутствие равных элементов. В рассматриваемом определении мы допускаем, что параметр n_k равен нулю. Например, $S_{210}(x_1, x_2, x_3) = x_1^2x_2 + x_1^2x_3 + x_1x_2^2 + x_1x_2x_3 + x_1x_2x_3 + x_1x_3^2 + x_2^2x_3 + x_2x_3^2$ для обобщенной диаграммы $\begin{smallmatrix}11\\2\end{smallmatrix}, \begin{smallmatrix}11\\3\end{smallmatrix}, \begin{smallmatrix}12\\2\end{smallmatrix}, \begin{smallmatrix}12\\3\end{smallmatrix}, \begin{smallmatrix}13\\2\end{smallmatrix}, \begin{smallmatrix}13\\3\end{smallmatrix}, \begin{smallmatrix}22\\3\end{smallmatrix}, \begin{smallmatrix}23\\3\end{smallmatrix}$. Количество таких диаграмм равно $\Delta(1,3,5)/\Delta(1,2,3) = 8$. Распространяя алгоритмы I и D на обобщенные диаграммы [*Pacific J. Math.* **34** (1970), 709–727], можно получить комбинаторное доказательство весьма примечательных тождеств:

$$\sum_{\lambda} S_\lambda(x_1, \ldots, x_m) S_\lambda(y_1, \ldots, y_n) = \prod_{i=1}^{m} \prod_{j=1}^{n} \frac{1}{1 - x_i y_j},$$

$$\sum_{\lambda} S_\lambda(x_1, \ldots, x_m) S_{\lambda^T}(y_1, \ldots, y_n) = \prod_{i=1}^{m} \prod_{j=1}^{n} (1 + x_i y_j).$$

Здесь суммирование выполняется по всем возможным формам λ, а λ^T обозначает транспонированную форму. Впервые эти тождества были найдены в работе D. E. Littlewood, *Proc. London Math. Soc.* (2) **40** (1936), 40–70, Theorem V.

Замечание. Отсюда, в частности, следует, что любое произведение последовательных биномиальных коэффициентов $\binom{a}{k}\binom{a+1}{k}\ldots\binom{a+l}{k}$ можно разделить на $\binom{k}{k}\binom{k+1}{k}\ldots\binom{k+l}{k}$, поскольку это отношение есть $\Delta(a+l, \ldots, a+1, a, k-1, \ldots, 1, 0)/\Delta(l, \ldots, 1, 0)$. Значение $\Delta(l, \ldots, 1, 0) = (l-1)! \ldots 1!\,0!$ иногда называют суперфакториалом.

34. Длина уголка есть также длина любого зигзагообразного пути от левой нижней ячейки уголка (x, y) до его правой верхней ячейки (x', y'). Докажем более сильный результат. Если существует уголок длиной $a + b$, то существует либо уголок длиной a, либо уголок длиной b. Рассмотрим ячейки $(x, y) = (x_1, y_1)$, (x_2, y_2), $\ldots$, $(x_{a+b}, y_{a+b}) = (x', y')$, которые обрамляют нижнюю часть формы диаграммы. Если $x_{a+1} = x_a$, то ячейка (x_a, y_1) имеет уголок длиной a; в противном случае (x_{a+1}, y_{a+b}) имеет уголок длиной b. [*См. Japanese J. Math.* **17** (1940), 165–184, 411–423. Накаяма первым рассмотрел уголки в процессе анализа групп перестановок и подошел очень близко к открытию теоремы H.]

35. В результате выполнения шагов G3–G5 на 1 уменьшается точно h_{ij} элементов массива p, если q_{ij} возросло, поскольку алгоритм отслеживает зигзагообразный путь от $p_{n'_j j}$ к p_{in_i}. В следующий раз выполнение этих же шагов начнется с большего значения j или будет

проходить выше либо на уровне того же зигзага, что и в предыдущий раз. Таким образом, массив q заполняется слева направо и снизу вверх; для того чтобы заставить процесс идти в обратном направлении, мы должны организовать движение слева направо и сверху вниз.

H1. [Инициализация.] Присвоить $p_{ij} \leftarrow 0$, $1 \le j \le n_i$ и $1 \le i \le n'_1$. Затем присвоить $i \leftarrow 1$ и $j \leftarrow n_1$.

H2. [Поиск ненулевой ячейки.] Если $q_{ij} > 0$, перейти к шагу H3. В противном случае, если $i < n'_j$, увеличить i на 1 и повторить этот шаг. В противном случае, если $j > 1$, уменьшить j на 1, присвоить $i \leftarrow 1$ и повторить этот шаг. В противном случае остановить процесс (массив q теперь обнулен).

H3. [Уменьшение q, подготовка к движению по зигзагу.] Уменьшить q_{ij} на 1 и присвоить $l \leftarrow i$, $k \leftarrow n_i$.

H4. [Переход вниз или влево.] Если $l < n'_k$ и $p_{lk} > p_{(l+1)k}$, увеличить l на 1 и вернуться к шагу H4. В противном случае, если $k > j$, уменьшить k на 1 и вернуться к шагу H4. В противном случае вернуться к шагу H2. ▌

Первый зигзаг для данного столбца j заканчивается приращением $p_{n'_j j}$, поскольку $p_{1j} \le \cdots \le p_{n'_j j}$ влечет за собой $p_{n'_j j} > 0$. Каждый последующий путь для столбца j проходит ниже или на том же уровне, что и предыдущий, так что он также закончится на $p_{n'_j j}$. Неравенства, которые используются в этом способе, свидетельствуют о том, что построенный алгоритм является обратным по отношению к описанному при постановке задачи. [*J. Combinatorial Theory* **A21** (1976), 216–221.]

36. (a) Коэффициент при z^m есть число решений $m = \sum h_{ij} q_{ij}$, так что можно распространить на данный случай результат предыдущего упражнения. (b) Если $a_1, \ldots, a_k$ — любые положительные целые числа, можно доказать, используя индукцию по k, что

$$[z^m]\, 1/(1-z)(1-z^{a_1})\ldots(1-z^{a_k}) = \binom{m}{k}/a_1\ldots a_k + O(m^{k-1}).$$

Число разбиений m на не более чем n частей равно, таким образом, $\binom{m}{n-1}/n! + O(m^{n-2})$ при фиксированном n, как следует из упр. 5.1.1–15. Это также асимптотическое число разбиений $m = p_1 + \cdots + p_n$, когда части *различны* — $p_1 > \cdots > p_n > 0$ (см. упр. 5.1.1–16). Таким образом, число обратных плоских разбиений асимптотически приближается к $N\binom{m}{n-1}/n! + O(m^{n-2})$, если имеется N диаграмм данной формы из n ячеек. Из п. (a) это также есть $\binom{m}{n-1}/\prod h_{ij} + O(m^{n-2})$. [*Studies in Applied Math.* **50** (1971), 167–188, 259–279.]

37. Плоское разбиение на прямоугольнике эквивалентно обратному плоскому разбиению, так что длины уголков дают нам производящую функцию $1/\prod_{i=1}^{r}\prod_{j=1}^{c}(1-z^{i+j-1})$ на прямоугольнике $r \times c$. Если положим $r, c \to \infty$, то получим ответ в очень элегантном виде: $1/(1-z)(1-z^2)^2(1-z^3)^3\ldots$. [Вывод принадлежит Мак-Магону и опубликован в *Philosophical Transactions* **211** (1912), 75–110, 345–373, но там он имел гораздо более сложную форму. Первым простое доказательство нашел Леонард Карлиц (Leonard Carlitz), *Acta Arithmetica* **13** (1967), 29–47.]

38. (a) Вероятность равна $1/n$ при $k = l = 1$; в противном случае, применяя индукцию по $k + l$, получим

$$\frac{nP(I \setminus \{i_0\},\, J) + nP(I,\, J \setminus \{j_0\})}{n\, d_{i_0 j_0}} = \frac{(d_{i_0 b} + d_{a j_0})/(n\, d_{i_0 b}\ldots d_{i_{k-1} b}\, d_{a j_0}\ldots d_{a j_{l-1}})}{d_{i_0 b} + d_{a j_0}}.$$

(b) Суммирование по всем I и J дает $n^{-1}(1+d_{1b}^{-1})\ldots(1+d_{(a-1)b}^{-1})\,(1+d_{a1}^{-1})\ldots(1+d_{a(b-1)}^{-1})$, что, как легко видеть, равно $f(T \setminus \{(a,b)\})/f(T)$.

(c) Суммирование по всем угловым ячейкам дает 1, поскольку каждый путь заканчивается в какой-либо угловой ячейке. Таким образом, $\sum f(T \setminus \{(a,b)\}) = f(T)$, а это

доказывает теорему H, если применить индукцию по n. Более того, если разместить n в угловой ячейке в конце случайного пути и повторить процесс на оставшихся $n-1$ ячейках, получится каждый вариант диаграммы с вероятностью $1/f(T)$. [*Advances in Math.* **31** (1979), 104–109.]

39. (a) $Q_{11}\ldots Q_{1n}$ будут иметь вид $b_1\ldots b_n$, т. е. будут представлять собой таблицу инверсий исходных перестановок $P_{11}\ldots P_{1n}$ (см. раздел 5.1.1).

(b) $Q_{11}\ldots Q_{n1}$ — таблица инверсий с обратным знаком $(-C_1)\ldots(-C_n)$ из упр. 5.1.1–7.

(c) Это условие, очевидно, предусматривается на шаге P3.

(d) $\binom{1\,4}{2\,3} \to \left(\binom{1\,3}{2\,4}, \binom{0\ -1}{0\ \ 0}\right)$; $\binom{4\,3}{1\,2} \to \left(\binom{1\,2}{3\,4}, \binom{0\ -1}{0\ \ 0}\right)$. Этот пример показывает, что нельзя выполнять шаг P3 в обратном направлении, не просмотрев массив P.

(e)

12	10	8	14	15	11
9	13	7	1		
6	4	5			
16	3				
2					

.

(f) Следующий алгоритм корректен, но не очевиден.

Q1. [Цикл по (i,j).] Выполнить шаги Q2 и Q3 для каждой ячейки (i,j) массива в лексикографическом порядке (т. е. сверху вниз и слева направо в каждой строке), затем прекратить выполнение процедуры.

Q2. [Изменение Q.] Найти "первого кандидата" (r,s) по правилу, которое будет изложено ниже. Затем присвоить $Q_{i(k+1)} \leftarrow Q_{ik} - 1$ для $j \le k < s$.

Q3. [Восстановление P в (i,j).] Присвоить $K \leftarrow P_{rs}$. Затем выполнять следующие операции до тех пор, пока не будет выполнено условие $(r,s) = (i,j)$. Если $P_{(r-1)s} > P_{r(s-1)}$, присвоить $P_{rs} \leftarrow P_{(r-1)s}$ и $r \leftarrow r-1$; в противном случае присвоить $P_{rs} \leftarrow P_{r(s-1)}$ и $s \leftarrow s-1$. И, наконец, присвоить $P_{ij} \leftarrow K$. ▌

На шаге Q2 ячейка (r,s) является *кандидатом*, если $s \ge j$ и $Q_{is} \le 0$, и $r = i - Q_{is}$. Пусть T — ориентированное дерево, построенное в соответствии с указанием. Один из основных инвариантов алгоритма Q состоит в том, что на этом дереве будет существовать путь от (r,s) к (i,j) в T всякий раз, когда (r,s) является кандидатом на шаге Q2. Обратный путь к нему можно закодировать последовательностью букв D, Q и R, означающих, что мы начали в ячейке (i,j), затем спустились (D) или пошли вправо (R), или закончили (Q). *Первый кандидат* — это один из кандидатов, код которого в лексикографическом смысле стоит раньше других; интуитивно кажется, что он должен быть крайним слева и снизу по отношению к "конкурентам".

Например, кандидатами при $(i,j) = (1,1)$ в примере п. (e) являются $(3,1)$, $(4,2)$, $(2,3)$, $(2,4)$ и $(1,6)$. Им соответствуют коды DDQ, DDDRQ, RDRQ, RDRRQ и RRRRRQ; из них первым будет $(4,2)$.

Алгоритм P представляет собой несколько упрощенную версию построения, описанного без доказательства в работе, опубликованной в журнале *Функциональный анализ и его приложения*, **26**, 3 (1992), 80–82. Доказательство корректности этого построения отнюдь не очевидно и дано Ж.-К. Новелли (J.-C. Novelli), И. Паком (I. Pak) и А. В. Стояновским (A. V. Stoyanovskii) в *Disc. Math. and Theoretical Comp. Sci.* **1** (1997), 53–67.

40. Эквивалентный процесс проанализирован в работе H. Rost, *Zeitschrift für Wahrscheinlichkeitstheorie und verwandte Gebiete* **58** (1981), 41–53.

41. (Решение предложено Р. У. Флойдом (R. W. Floyd).) Операция удаления-вставки фактически перемещает только a_i. В процессе ее выполнения незатронутые элементы сохраняют порядок. Таким образом, если перестановку π можно рассортировать при помощи k

операций удаления-вставки, то она имеет возрастающую подпоследовательность длиной $n-k$, и наоборот. Следовательно, $\mathrm{dis}(\pi) = n -$ (длина самой длинной возрастающей подпоследовательности в π) $= n +$ (длина строки 1 в теореме A).

М. Л. Фредман (M. L. Fredman) доказал, что наименьшее количество сравнений, необходимое для вычисления этой длины, равно $n \lg n - n \lg\lg n + O(n)$ [*Discrete Math.* **11** (1975), 29–35].

42. Постройте мультиграф с вершинами $\{0_R, 1_L, 1_R, \ldots, n_L, n_R, (n+1)_L\}$ и ребрами k_R — $(k+1)_L$ при $0 \le k \le n$, включите также ребра 0_R — 7_R, 7_L — 1_L, 1_R — 2_L, 2_R — 4_L, 4_R — 5_L, 5_R — 3_L, 3_R — 6_R, 6_L — 8_L, которые соответствуют "узам" в *Lobelia fervens*. К каждой вершине подходит в точности два ребра, так что связанные компоненты являются циклами: $(0_R\, 1_L\, 7_L\, 6_R\, 3_R\, 4_L\, 2_R\, 3_L\, 5_R\, 6_L\, 8_L\, 7_R)(1_R\, 2_L)(4_R\, 5_L)$. Некоторая операция перебрасывания изменяет количество циклов на -1, 0 или $+1$. Таким образом, нам понадобится, по крайней мере, пять операций для того, чтобы прийти к восьми циклам $(0_R\, 1_L)(1_R\, 2_L)\ldots(7_R\, 8_L)$. [J. Kececioglu, D. Sankoff, *Lecture Notes in Comp. Sci.* **807** (1994), 307–325.]

Первая операция должна разорвать узы 6_L — 8_L, поскольку мы не достигнем никакого нового цикла после разрыва двух уз, которые имеют одинаковую ориентацию слева направо в линейном представлении. В результате после одной операции появляется пять вариантов, а именно — $g_7^R g_6 g_3^R g_5^R g_4^R g_2^R g_1^R$, $g_7^R g_1 g_2 g_4 g_5 g_3 g_6$, $g_7^R g_1 g_2 g_6 g_3^R g_5^R g_4^R$, $g_7^R g_1 g_2 g_4 g_5 g_6 g_3^R$ и $g_6 g_3^R g_5^R g_4^R g_2^R g_1^R g_7$; еще четырех операций достаточно для того, чтобы рассортировать все, кроме второй из них.

Существует $2^7 \cdot 7! = 645\,120$ разных вариантов компоновки $g_1 \ldots g_7$ и 179 904 из них находятся на расстоянии ≤ 5 от порядка в молекуле ДНК табака.

[Эффективный алгоритм поиска наилучшего способа сортировки любой перестановки со знаком посредством реверсирования был изложен в работе S. Hannenhalli, P. Pevzner, *JACM* **46** (1999), 1–27, а усовершенствован — в работе Kaplan, Shamir, Tarjan, *SODA* **8** (1997), 344–351.]

43. Обозначим компоновку наподобие $g_7^R g_1 g_2 g_4 g_5 g_3 g_6^R$ посредством перестановки со знаком $\overline{7}12453\overline{6}$. Если существует отрицательный элемент, например $\overline{k}$, но отсутствует $\overline{k-1}$, одна операция перебрасывания создаст двойной цикл $((k-1)_R\, k_L)$. Аналогично, если имеется $\overline{k}$, но отсутствует $\overline{k+1}$, единственная операция перебрасывания сформирует $(k_R\, (k+1)_L)$. И если все операции такого вида удаляют все отрицательные элементы, то единственное перебрасывание формирует два двойных цикла. Если отсутствуют отрицательные элементы, а перестановка не рассортирована, некоторые из перебрасываний будут сохранять количество циклов. Следовательно, за $\le n$ перебрасываний можно выполнить сортировку, если данная перестановка имеет отрицательный элемент; в противном случае необходимое количество перебрасываний составит $\le n+1$.

Если n четное, перестановка $n\,(n-1)\ldots 1$ требует $n+1$ перебрасываний, поскольку в ней будет один цикл после первого перебрасывания. Если $n > 3$ нечетно, то, рассуждая аналогично, приходим к выводу, что для перестановки $2\,1\,3\,n\,(n-1)\ldots 4$ потребуется $n+1$ операция.

44. Пусть c_k — число циклов длиной $2k$ в мультиграфе из ответа к предыдущему упражнению. Верхняя граница для среднего значения c_k может быть найдена следующим образом. Общее число потенциальных $2k$-циклов равно $2^k (n+1)^{\underline{k}}/(2k)$, поскольку мы можем выбрать $(n+1)^{\underline{k}}$ способами последовательность k различных ребер из $\{0_R$ — $1_L, \ldots, n_R$ — $(n+1)_L\}$ и ориентировать их 2^k способами. Таким образом, каждый цикл будет подсчитан $2k$ раз, включая и невозможные случаи наподобие $(1_R\, 2_L\, 2_R\, 3_L)$, $(1_R\, 2_L\, 3_L\, 2_R\, 3_R\, 4_L)$ и $(1_R\, 2_L\, 6_R\, 7_L\, 4_L\, 3_R\, 2_R\, 3_L\, 6_L\, 5_R)$. Если $k \le n$, каждый возможный $2k$-цикл появляется точно в $2^{n-k}(n-k)!$ перестановках со знаком. Например, рассмотрим случай, когда $k = 5$,

$n = 9$ и имеется цикл $(0_R\, 1_L\, 9_L\, 8_R\, 7_R\, 8_L\, 1_R\, 2_L\, 5_L\, 4_R)$. Этот цикл появляется в мультиграфе тогда и только тогда, когда перестановка со знаком начинается с $\overline{4}$ и содержит подцепочки $\overline{9}18\overline{7}$ и $\overline{2}5$ или их реверсы. Все решения можно получить, если найти все перестановки со знаком множества $\{1, 2, 3, 6\}$ и заменить 1 значением $\overline{9}18\overline{7}$, 2 — значением $\overline{2}5$. Значит, $\mathrm{E}\, c_k \le 1/(2k)\, 2^k (n+1)^{\underline{k}} 2^{n-k} (n-k)!/2^n n! = \frac{1}{2}(1/k + 1/(n+1-k))$. Из этого следует, что $\mathrm{E}\, c = \sum_{k=1}^{n} \mathrm{E}\, c_k + \mathrm{E}\, c_{n+1} < H_n + 1$. Поскольку $n + 1 - c$ — нижняя граница числа перебрасываний, нам понадобится $\ge n + 1 - \mathrm{E}\, c > n - H_n$ из них.

[В этом доказательстве использована идея В. Бафна (V. Bafna) и П. Певзнера (P. Pevzner), *SICOMP* **25** (1996), 272–289, которые изучали более сложную проблему сортировки перестановки *без знака* посредством реверсирования. В этой задаче рассматривались перестановки, которые могут быть записаны в виде произведения неразрезанных циклов $(1\,2\,3)(3\,4\,5)(5\,6\,7)\ldots$, оканчивающихся либо на $(n{-}1\ n)$, либо на $(n{-}2\ n{-}1\ n)$ в зависимости от того, каким является n: четным или нечетным. Оказалось, что такие перестановки труднее всего сортировать.]

РАЗДЕЛ 5.2

1. Да; i и j могут изменяться в диапазоне $1 \le j < i \le N$ в произвольном порядке. Это позволяет выполнять подсчет параллельно с вводом данных.

2. Сортировка будет устойчива в том смысле, как определено в начале этой главы, поскольку алгоритм, по существу, выполняет упорядочение в лексикографическом порядке *различающихся* пар ключей $(K_1, 1)$, $(K_2, 2), \ldots, (K_N, N)$. (Если представить себе, что к каждому ключу справа добавлен его адрес в файле, то равных ключей не будет, а значит, сортировка будет устойчива.)

3. Программа все-таки будет выполнять сортировку, но это будет неустойчивая сортировка. Если $K_j = K_i$ и $j < i$, то, в конце концов, R_j окажется *после* R_i. Кроме того, внесенные изменения замедлят работу программы C.

4.
```
ENT1 N          1        STA  OUTPUT+1,2   N
LD2  COUNT,1    N        DEC1 1            N
LDA  INPUT,1    N        J1P  *-4          N  ▮
```

5. Время выполнения уменьшится на $A + 1 - N - B$ машинных циклов, и это почти всегда можно рассматривать как существенное улучшение.

6. $u = 0$, $v = 9$..

```
       После D1  COUNT =  0  0  0  0  0  0  0  0  0  0
       После D2  COUNT =  2  2  1  0  1  3  3  2  1  1
       После D4  COUNT =  2  4  5  5  6  9 12 14 15 16
   Во время D5   COUNT =  2  3  5  5  5  8  9 12 15 16        j = 8
                 OUTPUT = -- -- -- 1G -- 4A -- -- 5L 6A 6T 6I 7O 7N -- --
       После D5  OUTPUT = 0C 0O 1N 1G 2R 4A 5T 5U 5L 6A 6T 6I 7O 7N 8S  9
```

7. Да; обратите внимание на то, что COUNT[K_j] уменьшено на шаге D6, а затем уменьшается j.

8. Сортировка будет выполняться, но не будет устойчивой (см. упр. 7).

9. Пусть $M = v - u$; будем считать, что $|u|$ и $|v|$ умещаются в два байта. LOC(R_j) $\equiv$ INPUT $+ j$; LOC(COUNT[j]) $\equiv$ COUNT $+ j$; LOC(S_j) $\equiv$ OUTPUT $+ j$; rI1 $\equiv i$; rI2 $\equiv j$; rI3 $\equiv i - v$ или rI3 $\equiv K_j$.

```
M   EQU  V-U
KEY EQU  0:2                    (Сопутствующая информация в байтах 3:5)
```

```
1H  ENN3 M              1      D1. Очистка COUNT.
    STZ  COUNT+V,3      M+1    COUNT[v - k] ← 0.
    INC3 1              M+1
    J3NP *-2            M+1    u ≤ i ≤ v.
2H  ENT2 N              1      D2. Цикл j.
3H  LD3  INPUT,2(KEY)   N      D3. Увеличить COUNT[K_j].
    LDA  COUNT,3        N
    INCA 1              N
    STA  COUNT,3        N
    DEC2 1              N
    J2P  3B             N      N ≥ j > 0.
    ENN3 M-1            1      D4. Накопление.
    LDA  COUNT+U        1      rA ← COUNT[i - 1].
4H  ADD  COUNT+V,3      M      COUNT[i - 1] + COUNT[i]
    STA  COUNT+V,3      M         → COUNT[i].
    INC3 1              M
    J3NP 4B             M      u ≤ i ≤ v.
5H  ENT2 N              1      D5. Цикл j.
6H  LD3  INPUT,2(KEY)   N      D6. Вывод R_j.
    LD1  COUNT,3        N      i ← COUNT[K_j].
    LDA  INPUT,2        N      rA ← R_j.
    STA  OUTPUT,1       N      S_i ← rA.
    DEC1 1              N
    ST1  COUNT,3        N      COUNT[K_j] ← i - 1.
    DEC2 1              N
    J2P  6B             N      N ≥ j > 0.  ▌
```

Время выполнения — $(10M + 22N + 10)u$.

10. Чтобы не использовать дополнительно N бит для "маркера" [см. раздел 1.3.3 и *Cybernetics* **1** (1965), 95] и при этом все-таки оставить время выполнения пропорциональным N, можно использовать следующий алгоритм, основанный на циклической структуре перестановки.

P1. [Цикл по i.] Выполнить шаг P2 для $1 \le i \le N$; затем завершить выполнение процедуры.

P2. [$p(i) = i$?] Выполнить шаги с P3 по P5, если $p(i) \ne i$.

P3. [Начало цикла.] Присвоить $t \leftarrow R_i$, $j \leftarrow i$.

P4. [Обработать R_j.] Присвоить $k \leftarrow p(j)$, $R_j \leftarrow R_k$, $p(j) \leftarrow j$, $j \leftarrow k$. Если $p(j) \ne i$, повторить этот шаг.

P5. [Конец цикла.] Присвоить $R_j \leftarrow t$, $p(j) \leftarrow j$. ▌

Этот алгоритм изменяет $p(i)$, поскольку в приложениях, использующих сортировку, можно считать, что $p(i)$ хранится в оперативной памяти. С другой стороны, существуют приложения, такие как транспонирование матриц, в которых $p(i)$ является функцией i, т. е. его для экономии памяти необходимо вычислять, а не считывать из таблицы. В этом случае можно использовать следующий метод, выполняя шаги от B1 до B3 для $1 \le i \le N$.

B1. Присвоить $k \leftarrow p(i)$.

B2. Если $k > i$, присвоить $k \leftarrow p(k)$ и повторить этот шаг.

B3. Если $k < i$, ничего не выполнять, но если $k = i$ (это означает, что i — наименьшее в своем цикле), то переставить операции цикла, связанные с i следующим образом.

Присвоить $t \leftarrow R_i$; затем, пока $p(k) \neq i$, повторять присвоения $R_k \leftarrow R_{p(k)}$ и $k \leftarrow p(k)$; и, наконец, присвоить $R_k \leftarrow t$. ▌

Этот алгоритм аналогичен процедуре, описанной в работе J. Boothroyd, *Comp. J.* **10** (1967), 310,но требует меньшего количества операций перемещения данных; несколько усовершенствованный алгоритм предложил И. Д. Г. Мак-Леод (I. D. G. MacLeod) [*Australian Comp. J.* **2** (1970), 16–19]. Анализ, выполненный для случайной перестановки в упр. 1.3.3–14, показывает, что шаг B2 выполняется в среднем $(N+1)H_N - N$ раз. (См. также ссылки на литературу в ответе к упр. 1.3.3–12.) Аналогичный алгоритм можно разработать и для того, чтобы заменить $(R_{p(1)}, \ldots, R_{p(N)})$ на $(R_1, \ldots, R_N)$, например, если перекомпоновка в упр. 4 должна быть выполнена при условии OUTPUT = INPUT.

11. Пусть rI1 $\equiv i$; rI2 $\equiv j$; rI3 $\equiv k$; rX $\equiv t$.

1H ENT1	N	1	*P1. Цикл по i.*
2H CMP1	P,1	N	*P2. $p(i) = i$?*
JE	8F	N	Перейти, если $p(i) = i$.
3H LDX	INPUT,1	$A-B$	*P3. Начать цикл.* $t \leftarrow R_i$.
ENT2	0,1	$A-B$	$j \leftarrow i$.
4H LD3	P,2	$N-A$	*P4. Зафиксировать R_j.* $k \leftarrow p(j)$.
LDA	INPUT,3	$N-A$	
STA	INPUT,2	$N-A$	$R_j \leftarrow R_k$.
ST2	P,2	$N-A$	$p(j) \leftarrow j$.
ENT2	0,3	$N-A$	$j \leftarrow k$.
CMP1	P,2	$N-A$	
JNE	4B	$N-A$	Повторить, если $p(j) \neq i$.
5H STX	INPUT,2	$A-B$	*P5. Конец цикла.* $R_j \leftarrow t$.
ST2	P,2	$A-B$	$p(j) \leftarrow j$.
8H DEC1	1	N	
J1P	2B	N	$N \geq i \geq 1$. ▌

Время выполнения равно $(17N - 5A - 7B + 1)u$, где A — число циклов в перестановке $p(1)\ldots p(N)$ и B — число неподвижных точек перестановки (единичных циклов). Получим

$$A = (\min 1,\ \operatorname{ave} H_N,\ \max N,\ \operatorname{dev}\sqrt{H_N - H_N^{(2)}}\,) \text{ и } B = (\min 0,\ \operatorname{ave} 1,\ \max N,\ \operatorname{dev} 1)$$

для $N \geq 2$ в соответствии с 1.3.3–(21) и 1.3.3–(28).

12. Очевидный способ — пройти по всему списку, заменяя связи k-го элемента числом k, и затем перекомпоновать элементы на втором проходе. Описанный ниже более прямой и быстрый способ для случая, когда записи не слишком велики, принадлежит М. Д. Мак-Ларену (M. D. MacLaren). (Для удобства предполагается, что $0 \leq$ LINK(P) $\leq N$ при $1 \leq$ P $\leq N$, где $\Lambda \equiv 0$.)

M1. [Инициализация.] Присвоить P $\leftarrow$ HEAD, $k \leftarrow 1$.

M2. [Выполнено?] Если P $= \Lambda$ (или, что равносильно, если $k = N+1$), выполнение процедуры заканчивается.

M3. [Обеспечить P $\geq k$.] Если P $< k$, присвоить P $\leftarrow$ LINK(P) и повторить этот шаг.

M4. [Обмен.] Поменять местами R_k и R[P]. (Предполагается, что LINK(k) и LINK(P) также при этом меняются местами.) Затем присвоить Q $\leftarrow$ LINK(k), LINK(k) $\leftarrow$ P, P $\leftarrow$ Q, $k \leftarrow k+1$ и вернуться к шагу M2. ▌

Доказательство состоятельности метода М. Д. Мак-Ларена может базироваться на проверке по индукции следующего свойства, которое всегда выполняется перед началом шага M2: элементы, которые $\geq k$ в последовательности P, LINK(P), LINK(LINK(P)), ..., Λ, — это a_1,

$a_2, \ldots, a_{N+1-k}$, где $R_1 \le \cdots \le R_{k-1} \le R_{a_1} \le \cdots \le R_{a_{N+1-k}}$ — требуемый окончательный порядок записей. Далее, LINK(j) $\ge j$ для $1 \le j < k$, так что из равенства LINK(j) $= \Lambda$ следует $j \ge k$.

Довольно интересно проанализировать алгоритм М. Д. Мак-Ларена. Одно из его замечательных свойств состоит в том, что алгоритм можно выполнить в обратном порядке и восстановить исходное множество связей из конечных значений LINK(1) ... LINK(N). Каждая из $N!$ возможных конфигураций на выходе при $j \le$ LINK(j) $\le N$ соответствует в точности одной из $N!$ возможных конфигураций на входе. Если обозначить через A количество операций P $\leftarrow$ LINK(P) на шаге M3, то $N - A$ — число индексов j, таких, что LINK(j) $= j$ после завершения выполнения процедуры. Это возникает тогда и только тогда, когда j наибольшее в своем цикле; следовательно, $N - A$ равно числу циклов перестановки, а $A = (\min 0,\ \text{ave}\ N - H_N,\ \max N - 1)$.

См. M. D. MacLaren, *JACM* **13** (1966), 404–411; D. Gries, J. F. Prins, *Science of Computer Programming* **8** (1987), 139–145.

13. **D5′.** Присвоить $r \leftarrow N$.

D6′. Если $r = 0$, остановиться. В противном случае, если COUNT[K_r] $< r$, присвоить $r \leftarrow r - 1$ и повторить этот шаг; если COUNT[K_r] $= r$, уменьшить и COUNT[K_r], и r на 1 и повторить этот шаг. В противном случае присвоить $R \leftarrow R_r$, $j \leftarrow$ COUNT[K_r], COUNT[K_r] $\leftarrow j - 1$.

D7′. Присвоить $S \leftarrow R_j$, $k \leftarrow$ COUNT[K_j], COUNT[K_j] $\leftarrow k - 1$, $R_j \leftarrow R$, $R \leftarrow S$, $j \leftarrow k$. Затем, если $j \ne r$, повторить этот шаг; если $j = r$, присвоить $R_j \leftarrow R$, $r \leftarrow r - 1$ и вернуться к шагу D6′. ▌

Для того чтобы убедиться в состоятельности этой процедуры, обратите внимание, что перед началом шага D6′ все записи R_j, которые еще не находятся на своих окончательных местах и для которых $j > r$, должны продвинуться влево; когда $r = 0$, не может существовать ни одна такая запись, поскольку *нечему* двигаться вправо. Алгоритм, конечно, очень элегантен, но, к сожалению, нестабилен при наличии равных ключей. Он тесно связан с построением Фоаты в теореме 5.1.2B.

РАЗДЕЛ 5.2.1

1. Да. Равные элементы никогда не меняются местами.

2. Да, но в случае, если имеются равные элементы, время работы увеличится и процесс сортировки будет выполняться в прямо противоположном направлении по сравнению с устойчивой сортировкой.

3. Предполагается, что следующая программа из 8 команд — самая короткая программа сортировки для машины MIX, хотя она и не может быть рекомендована из-за низкой скорости выполнения. Считается, что числа находятся в ячейках $1, \ldots, N$ (т. е. INPUT EQU 0); в противном случае нужна еще одна команда.

```
2H     LDA  0,1    B
       CMPA 1,1    B
       JLE  1F     B
       MOVE 1,1    A
       STA  0,1    A
START  ENT1 N      A+1
1H     DEC1 1      B+1
       J1P  2B     B+1  ▌
```

Замечание. Чтобы оценить время выполнения, заметим, что A равно числу инверсий, а величина B — довольно простая функция таблицы инверсий и (в предположении, что

элементы ввода различны и расположены в случайном порядке) имеет производящую функцию

$$z^{N-1}(1+z)(1+z^2+z^{2+1})$$
$$\times(1+z^3+z^{3+2}+z^{3+2+1})\ldots(1+z^{N-1}+z^{2N-3}+\cdots+z^{N(N-1)/2})/N!\,.$$

Среднее значение B равно $N-1+\sum_{k=1}^{N}(k-1)(2k-1)/6=(N-1)(4N^2+N+36)/36$; следовательно, среднее время выполнения — примерно $\frac{7}{9}N^3u$.

4. Рассмотрим таблицу инверсий $B_1\ldots B_N$ данной исходной перестановки в смысле упр. 5.1.1–7. Величина A на единицу меньше числа элементов B_j, которое равно $j-1$, а B равно сумме элементов B_j. Следовательно, обе величины — $B-A$ и B — достигают максимума, когда исходная перестановка равна $N\ldots 2\ 1$; обе величины достигают минимума, когда исходная перестановка равна $1\ 2\ldots N$. Следовательно, минимальное возможное время выполнения достигается при $A=0$ и $B=0$ и равно $(10N-9)u$; максимальное время выполнения будет достигнуто при $A=N-1$ и $B=\binom{N}{2}$ и равно $(4.5N^2+2.5N-6)u$.

5. Искомая производящая функция равна произведению z^{10N-9} и производящей функции для величины $9B-3A$. Рассмотрев, как в предыдущем упражнении, таблицу инверсий и вспомнив, что отдельные элементы таблицы инверсий не зависят один от другого, находим искомую производящую функцию $z^{10N-9}\prod_{1<j\le N}((1+z^9+\cdots+z^{9j-18}+z^{9j-12})/j)$. Дисперсия равна $2.25N^3+3.375N^2-32.625N+36H_N-9H_N^{(2)}$.

6. Организуйте область вывода как циклический список, в котором позиция N будет соседней по отношению к позиции 1. Следующий элемент, который нужно вставить, берется с левого или правого конца текущей серии нерассортированных элементов в зависимости от того, оказался ли предыдущий вставленный элемент соответственно справа или слева от центра области рассортированных элементов. В конце обычно требуется "прокрутить" эту область, переместив каждую запись на k позиций по кругу, где k — некоторое фиксированное значение. Это можно эффективно выполнить способом, аналогичным рассмотренному в упр. 1.3.3–34.

7. Среднее значение для $|a_j-j|$ равно

$$\frac{1}{n}(|1-j|+|2-j|+\cdots+|n-j|)=\frac{1}{n}\left(\binom{j}{2}+\binom{n-j+1}{2}\right);$$

суммируя по j, получим $\frac{1}{n}(\binom{n+1}{3}+\binom{n+1}{3})=\frac{1}{3}(n^2-1)$.

8. Нет; рассмотрите, например, последовательность ключей 2 1 1 1 1 1 1 1 1 1 1.

9. Для табл. 3 $A=3+0+2+1=6$, $B=3+1+4+21=29$; для табл. 4 $A=4+2+2+0=8$, $B=4+3+8+10=25$; следовательно, время выполнения программы D в этих двух случаях равно соответственно $786u$ и $734u$. Хотя число перезаписей сократилось с 41 до 25, эта программа не может соперничать с программой S по времени работы, поскольку при $N=16$ тратится много времени на вспомогательные операции, необходимые для организации четырех проходов. При сортировке 16 элементов лучше выполнить только два просмотра: двухпроходный вариант программы D начинает превосходить по скорости программу S примерно при $N=13$; и все же на коротких наборах входных данных они почти равноценны (а при малых значениях N, возможно, существенную роль играет длина программы).

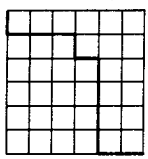
10. Вставить "`INC1 INPUT; ST1 3F(0:2)`" между командами в строках 07 и 08 и заменить строки 10–17 следующими строками:

```
3H CMPA INPUT+N-H,1      NT − S
   JGE  8F               NT − S
```

```
4H ENT2 N-H,1          NT − S − C
5H LDX  INPUT,2            B
6H STX  INPUT+H,2          B
   DEC2 0,4                B
   J2NP 7F                 B
   CMPA INPUT,2          B − A
   JL   5B               B − A
7H STA  INPUT+H,2     NT − S − C   ▌
```

За счет увеличения программы на четыре команды удается сэкономить $3(C-T)$ машинных циклов, где C — число случаев, когда $K_j \geq K_{j-h}$. В табл. 3 и 4 экономия времени составляет соответственно около 87 и 88 циклов; эмпирически значение $C/(NT-S)$ можно выбрать равным приблизительно 0.4, если $h_{s+1}/h_s \approx 2$ и приблизительно 0.3, если $h_{s+1}/h_s \approx 3$, так что это усовершенствование стоит затраченных усилий. (С другой стороны, аналогичное изменение программы S нежелательно, так как экономия в этом случае пропорциональна всего лишь $\log N$, если только заранее неизвестно, что входные данные достаточно хорошо упорядочены.)

11.



12. Замена символом всегда приводит к изменению количества инверсий на ± 1 в зависимости от того, где выполнена замена — над или под диагональю.

13. Припишите вес $|i-j|$ сегменту от $(i, j-1)$ до (i, j).

14. (a) Поменяйте местами i и j в сумме для A_{2n} и сложите эти две суммы. (b) Взяв половину данного результата, видим, что

$$A_{2n} = \sum_{0 \leq i \leq j} (j-i)\binom{i+j}{i}\binom{2n-i-j}{n-j} = \sum_{i,k \geq 0} k\binom{2i+k}{i}\binom{2n-2i-k}{n-i-k};$$

следовательно, $\sum A_{2n} z^n = \sum_{k \geq 0} k z^k \alpha^{2k}/(1-4z) = z/(1-4z)^2$, где $\alpha = (1-\sqrt{1-4z})/2z$.

Это доказательство сообщил автору Леонард Карлиц (Leonard Carlitz). Еще одно доказательство может основываться на взаимосвязи между весами для горизонтальных и вертикальных серий (см. упр. 13). Другой вариант доказательства можно сформулировать при помощи тождества, которое рассмотрено в ответе к упр. 5.2.2–16 при $f(k) = k$; однако непонятно, как, используя комбинаторику, просто вывести формулу $A_n = \lfloor n/2 \rfloor 2^{n-2}$.

15. При $n > 0$

$$\hat{g}_n(z) = z^n g_{n-1}; \qquad \hat{h}_n(z) = \hat{g}_n(z) + z^{-n}\,\hat{g}_n(z);$$

$$g_n(z) = \sum_{k=1}^{n} \hat{g}_k(z) g_{n-k}(z); \qquad h_n(z) = \sum_{k=1}^{n} \hat{h}_k(z) h_{n-k}(z).$$

Полагая $G(w,z) = \sum_n g_n(z) w^n$, находим, что $wzG(w,z)G(wz,z) = G(w,z) - 1$. Из этого представления можно вывести, что, если $t = \sqrt{1-4w} = 1 - 2w - 2w^2 - 4w^3 - \cdots$, имеем $G(w,1) = (1-t)/(2w)$; $G_t(w,1) = 1/(wt) - (1-t)/(2w^2)$; $G'(w,1) = 1/(2t^2) - 1/(2t)$; $G_{tt}(w,1) = 2/(wt^3) - 2/(w^2 t) + (1-t)/w^3$; $G'_t(w,1) = 2/t^4 - 1/t^3$; $G''(w,1) = 1/t^3 - (1-2w)/t^4 + 10w^2/t^5$. Здесь нижние штрихи обозначают дифференцирование по первому параметру, а верхние — по второму параметру. Аналогично из формулы

$$w(zG(wz,z) + G(w,z))H(w,z) = H(w,z) - 1$$

получим

$$H'(w,1) = w/t^4, \qquad H''(w,1) = -w/t^3 - w/t^4 + 2w/t^5 + (2w^2 + 20w^3)/t^7.$$

Все эти манипуляции формулами выполнены вручную, но современные программные средства позволяют проделать то же самое значительно быстрее на компьютере. В принципе, таким способом можно получить все моменты этого распределения.

Производящая функция $g_n(z)$ также представляет $\sum z^{\text{длина внутреннего пути}}$ по всем деревьям с $n+1$ узлами (см. упр. 2.3.4.5–5). Интересно отметить, что $G(w,z)$ равна $F(-wz,z)/F(-w,z)$, где $F(z,q) = \sum_{n\ge 0} z^n q^{n^2}/\prod_{k=1}^n (1-q^k)$; коэффициент при $q^m z^n$ в $F(z,q)$ равен числу разбиений $m = p_1 + \cdots + p_n$, таких, что $p_j \ge p_{j+1} + 2$ при $1 \le j < n$ и $p_n > 0$ (см. упр. 5.1.1–16).

16. Ясно, что при $h = 2$ максимум достигается на пути, проходящем через правый верхний угол решеточной диаграммы, и равен

$$\binom{\lfloor n/2 \rfloor + 1}{2}.$$

При произвольном значении h соответствующее число равно

$$\hat f(n,h) = \binom{h}{2}\binom{q+1}{2} + \binom{r}{2}(q+1),$$

где q и r определяются теоремой H; перестановка, в которой

$$a_{i+jh} = 1 + q(h-i) + (r-i)[i \le r] \qquad \text{при } 1 \le i \le h \text{ и } j \ge 0,$$

максимизирует число инверсий в каждой из $\binom{h}{2}$ пар упорядоченных подпоследовательностей. Максимальное число перемещений получится, если в формуле (6) подставить $\hat f$ вместо f.

17. Единственная 2-упорядоченная перестановка множества $\{1, 2, \ldots, 2n\}$ с $\binom{n+1}{2}$ инверсиями — это $n{+}1\ 1\ n{+}2\ 2\ \ldots\ 2n\ n$. Применяя данную идею рекурсивно, получим перестановку, в которой добавлена единица к каждому элементу последовательности $(2^t-1)^R \ldots 1^R 0^R$, где R обозначает операцию записи целого в виде t-разрядного двоичного числа с последующей записью его двоичных разрядов в обратном порядке (справа налево)!

18. Вынесем за скобки общий множитель и положим $h_t = 4N/\pi$; требуется минимизировать сумму $\sum_{s=1}^t h_s^{1/2}/h_{s-1}$ при условии, что $h_0 = 1$. В результате дифференцирования получается соотношение $h_s^3 = 4h_{s-1}^2 h_{s+1}$, которое имеет решение $(2^t-1)\lg h_1 = 2^{t+1} - 2(t+1) + \lg h_t$. Минимальное значение исходной оценки равно $(1-2^{-t})\pi^{(2^{t-1}-1)/(2^t-1)} N^{1+2^{t-1}/(2^t-1)}/ /2^{1+(t-1)/(2^t-1)}$, при $t \to \infty$ эта величина быстро сходится к $N\sqrt{\pi N}/2$.

Ниже приведены типичные "оптимальные" значения h при $N = 1000$ (см. также табл. 6):

$$h_2 \approx 57.64, \qquad h_1 \approx 6.13, \qquad h_0 = 1;$$

$$h_3 \approx 135.30, \qquad h_2 \approx 22.05, \qquad h_1 \approx 4.45, \qquad h_0 = 1;$$

$$h_4 \approx 284.46, \qquad h_3 \approx 67.23, \qquad h_2 \approx 16.34, \qquad h_1 \approx 4.03, \qquad h_0 = 1;$$

$$h_9 \approx 9164.74, \quad h_8 \approx 12294.05, \quad h_7 \approx 7119.55, \quad h_6 \approx 2708.95, \quad h_5 \approx 835.50,$$
$$h_4 \approx 232.00, \quad h_3 \approx 61.13, \quad h_2 \approx 15.69, \quad h_1 \approx 3.97, \quad h_0 = 1.$$

19. Пусть $g(n,h) = H_r - 1 + \sum_{r<j\le h} q/(qj+r)$, где q и r определены в теореме H. Подставьте g вместо f в формуле (6).

20. (Сформулировать это на бумаге труднее, чем объяснить на пальцах.) Предположим, что k-упорядоченный массив $R_1, \ldots, R_N$ был h-рассортирован, и пусть $1 \le i \le N-k$; наша цель — показать, что $K_i \le K_{i+k}$. Найдем u, v, такие, что $i \equiv u$ и $i+k \equiv v$ (по модулю h), $1 \le u, v \le h$. Применим теперь лемму L при $x_j = K_{v+(j-1)h}$, $y_j = K_{u+(j-1)h}$. Затем первые r элементов K_u, K_{u+h}, $\ldots$, $K_{u+(r-1)h}$ из y_k будут соответственно $\le$ последним r элементам K_{u+k}, K_{u+k+h}, $\ldots$, $K_{u+k+(r-1)h}$ из x_k, где r — наибольшее целое число, такое, что $u+k+(r-1)h \le N$.

21. Если $xh+yk = x'h+y'k$, имеем $(x-x')h = (y'-y)k$, так что $x' = x+tk$ и $y' = y-th$ для некоторых целых t. Пусть $h'h+k'k = 1$; тогда $n = (nh')h + (nk')k$, так что любое целое n имеет единственное представление в виде $n = xh+yk$, где $0 \le x < k$, а n — порождаемое тогда и только тогда, когда $y \ge 0$. Пусть аналогично $hk-h-k-n = x'h+y'k$; тогда $(x+x')h+(y+y')k = hk-h-k$. Следовательно, $x+x' \equiv k-1$ (по модулю k) и должно быть $x+x' = k-1$. Отсюда $y+y' = -1$ и $y \ge 0$ тогда и только тогда, когда $y' < 0$.

Симметричность этого результата свидетельствует о том, что точно $\frac{1}{2}(h-1)(k-1)$ положительных целых можно представить в предлагаемом виде. Этот результат впервые получен Сильвестром (Sylvester) [*Mathematical Questions, with their Solutions, from the 'Educational Times'* **41** (1884), 21].

22. Чтобы избежать громоздких формул, рассмотрим $s = 4$ в качестве "полномочного" представителя общего случая. Пусть n_k — наименьшее число, которое можно представить в виде $15a_0 + 31a_1 + \cdots$ и конгруэнтное k (по модулю 15). Тогда нетрудно подсчитать, что

$$\begin{array}{rcccccccccccccccc} k &=& 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14, \\ n_k &=& 0 & 31 & 62 & 63 & 94 & 125 & 126 & 127 & 158 & 189 & 190 & 221 & 252 & 253 & 254. \end{array}$$

Следовательно, $239 = 2^4(2^4-1)-1$ — наибольшее среди чисел, которые нельзя представить в таком виде, а суммарное количество таких чисел есть

$$\begin{aligned} x_4 &= (n_1 - 1 + n_2 - 2 + \cdots + n_{14} - 14)/15 \\ &= (2+4+4+6+8+8)+8+(10+12+12+14+16+16)+16 \\ &= 2x_3 + 8 \cdot 9. \end{aligned}$$

В общем случае $x_s = 2x_{s-1} + 2^{s-1}(2^{s-1}+1)$.

Ответы на другие вопросы — соответственно $2^{2s}+2^s+2$ и $2^{s-1}(2^s+s-1)+2$.

23. Каждое из N чисел имеет не более $\lceil (h_{s+2}-1)(h_{s+1}-1)/h_s \rceil$ инверсий в своем подмассиве.

24. (Решение получено совместно с В. Праттом (V. Pratt). Построим h-возвратную перестановку множества $\{1, 2, \ldots, N\}$ следующим образом. Начав с пустых позиций $a_1 \ldots a_N$, выполним при $j = 2, 3, 4, \ldots$ шаг j. Слева направо заполняем пустые позиции a_i, используя наименьшее число, которое еще не появилось в перестановке, всякий раз, когда $(2^h-1)j-i$ есть положительное целое число, которое можно представить в виде, описанном в упр. 22. Процесс продолжается до тех пор, пока не будут заполнены все позиции. Так, 2-возвратной перестановкой при $N = 20$ будет

$$6\ 2\ 1\ 9\ 4\ 3\ 12\ 7\ 5\ 15\ 10\ 8\ 17\ 13\ 11\ 19\ 16\ 14\ 20\ 18.$$

При всех $k \ge h$ h-возвратная перестановка (2^k-1)-упорядочена. Если $2^h < j \le N/(2^h-1)$, то на j-м шаге заполняется ровно 2^h-1 позиций, причем $(k+1)$-я из них добавляет, по крайней мере, $2^{h-1}-2k$ к числу перемещений записей, требуемых для $(2^{h-1}-1)$-сортировки этой перестановки. Следовательно, число перемещений, необходимых для сортировки h-возвратной перестановки со смещениями $h_s = 2^s-1$ при $N = 2^{h+1}(2^h-1)$, заведомо больше $2^{3h-4} > \frac{1}{64}N^{3/2}$. В. Пратт обобщил это построение для обширного семейства аналогичных

последовательностей, включая (12), в своей докторской диссертации (Stanford University, 1972). Некоторые эвристические методы, позволяющие найти такие перестановки, которые нуждаются даже в большем числе перезаписей, найдены Х. Эркио (H. Erkiö), *BIT* **20** (1980), 130–136. См. также Weiss, Sedgewick, *J. Algorithms* **11** (1990), 242–251, где предлагается дальнейшее усовершенствование построения Пратта.

25. F_{N+1} [этот результат получен Г. Б. Манном (H. B. Mann), *Econometrica* **13** (1945), 256], так как перестановка должна начинаться либо с 1, либо с 2 1. Имеется не более $\lfloor N/2 \rfloor$ инверсий; общее число инверсий равно

$$\frac{N-1}{5} F_N + \frac{2N}{5} F_{N-1}.$$

(См. упр. 1.2.8–12.) Обратите внимание на то, что F_{N+1} перестановок можно удобно представить азбукой Морзе (последовательностью точек и тире), где тире соответствует инверсии (см. упр. 4.5.3–32). Следовательно, мы нашли общее число тире во всех последовательностях точек и тире, имеющих длину N.

Приведенные выкладки показывают, что случайная 3- и 2-упорядоченная перестановка имеет грубо $\frac{1}{5}(\phi^{-1}+2\phi^{-2})N = \phi^{-1}N/\sqrt{5} \approx .276N$ инверсий. Но если случайная перестановка является 3-рассортированной, а затем 2-рассортированной, то, как показано в упр. 42, она имеет $\approx N/4$ инверсий; если же сначала она является 2-рассортированной, а затем — 3-рассортированной, то она имеет $\approx N/3$ инверсий.

26. Да: пример самого короткого из них — 4 1 3 7 2 6 8 5. Здесь имеется 9 инверсий. В общем случае конструкция $a_{3k+s} = 3k + 4s$ при $-1 \le s \le 1$ дает 3-, 5- и 7-упорядоченные массивы, которые имеют приблизительно $\frac{4}{3}N$ инверсий. Когда $N \bmod 3 = 2$, эта конструкция — наилучшая из возможных.

27. (a) См. *J. Algorithms* **15** (1993), 101–124. Более простое доказательство, которое показывает, что c может быть любой константой $< \frac{1}{2}$, было независимо получено Ч. Г. Плакстоном (C. G. Plaxton) и Т. Сьюэлом (T. Suel), *J. Algorithms* **23** (1997), 221–240. (b) Это очевидно, если $m > \frac{1}{4}c^2(\ln N/\ln\ln N)^2$. В противном случае $N^{1+c/\sqrt{m}} \ge N(\ln N)^2$. Р. Э. Кифер (R. E. Cypher) [*SICOMP* **22** (1993), 62–71] нашел более строгое ограничение $\Omega(N(\log N)^2/\log\log N)$ для случая, если смещения удовлетворяют неравенству $h_{s+1} > h_s$ при всех s и если последовательность сортировки формируется так, как описано в упр. 5.3.4–2. На сегодняшний день неизвестно нетривиальное выражение нижнего асимптотического предела для среднего значения *времени* выполнения.

28. 209 109 41 19 5 1, как следует из (11). Но возможна и лучшая последовательность смещений (см. упр. 29).

29. В результате экспериментов Ш. Триболе (C. Tribolet) получил в 1971 году последовательности 373 137 53 19 7 3 1 ($B_{\text{ave}} \approx 7210$) и 317 101 31 11 3 1 ($B_{\text{ave}} \approx 8170$). [В первой из них время сортировки равно $\approx 127720u$ по сравнению с $\approx 128593u$, когда те же данные сортировались с последовательностью смещений (11).] В общем случае Триболе предлагает выбирать h_s равными простому числу, ближайшему к $N^{s/t}$. Эксперименты, проведенные в 1972 году Шелби Седжелом (Shelby Siegel), показывают, что наилучшее число смещений в последовательности для $N \le 10000$ при таком методе выбора последовательности равно $t \approx \frac{4}{3}\ln(N/5.75)$.

Еще одна хорошая последовательность, найденная Робертом Л. Томлинсоном (мл.) (Robert L. Tomlinson, Jr.), — 199 79 31 11 5 1 ($B_{\text{ave}} \approx 7950$). Среднее время в этом варианте — $\approx 127260u$ — оказалось лучшим из известных на сегодняшний день результатов.

Как следует из длительных экспериментов, проведенных Кэрол М. Мак-Нэйми (Carole M. McNamee), наилучшей последовательностью из трех смешений будет 45 7 1 ($B_{\text{ave}} \approx 18240$). Победительницей в "классе четырех смещений", как показали ее эксперименты,

оказалась последовательность 91 23 7 1 ($B_{\text{ave}} \approx 11865$), однако в довольно широком диапазоне значений смещений результаты получаются примерно одинаковые.

30. Число точек с целыми координатами в треугольной области

$$\{x \ln 2 + y \ln 3 < \ln N,\ x \geq 0,\ y \geq 0\} \quad \text{равно} \quad \tfrac{1}{2}(\log_2 N)(\log_3 N) + O(\log N).$$

Во время h-сортировки массив уже $2h$- и $3h$-упорядочен (см. теорему K); следовательно, можно использовать результат упр. 25.

31.

```
01  START ENT3 T                  1
02  1H    LD4  H,3                T
03        ENN2 -INPUT-N,4         T
04        ST2  6F(0:2)            T
05        ST2  7F(0:2)            T
06        ST2  4F(0:2)            T
07        ENT2 0,4                T
08        JMP  9F                 T
09  2H    LDA  INPUT+N,1          NT - S - B + A
10  4H    CMPA INPUT+N-H,1        NT - S - B + A
11        JGE  8F                 NT - S - B + A
12  6H    LDX  INPUT+N-H,1        B
13        STX  INPUT+N,1          B
14  7H    STA  INPUT+N-H,1        B
15        INC1 0,4                B
16  8H    INC1 0,4                NT - B + A
17        J1NP 2B                 NT - B + A
18        DEC2 1                  S
19  9H    ENT1 -N,2               T + S
20        J2P  8B                 T + S
21        DEC3 1                  T
22        J3P  1B                 T  ▌
```

Здесь A — число правосторонних максимумов, как A в программе D — число левосторонних минимумов; обе величины статистически ведут себя одинаково. В результате упрощения внутреннего цикла время выполнения сократилось до $7NT + 7A - 2S + 1 + 15T$ машинных циклов и, как ни странно, оно не зависит от B!

При $N = 8$ смещениями будут 6, 4, 3, 2, 1, и имеем $A_{\text{ave}} = 3.892$, $B_{\text{ave}} = 6.762$; суммарное время работы в среднем составляет $276.24u$ (ср. с табл. 5). Обе величины — A и B — достигают максимума на перестановке 7 3 8 4 5 1 6 2. При $N = 1000$ имеется последовательность из 40 смещений: $972, 864, 768, 729, \ldots, 8, 6, 4, 3, 2, 1$; эмпирические тесты, подобные тем, которые приведены в табл. 6, дают $A \approx 875$, $B \approx 4250$ и общее время выполнения — около $268000u$ (примерно вдвое больше, чем для программы D с последовательностью смещений из упр. 28).

Вместо того чтобы хранить значения смещений во вспомогательной таблице, удобно генерировать их на двоичной машине следующим образом.

P1. Присвоить $m \leftarrow 2^{\lceil \lg N \rceil - 1}$, наибольшая степень двойки, меньшая N.

P2. Присвоить $h \leftarrow m$.

P3. Использовать h в качестве смещения на очередном проходе сортировки.

P4. Если h четное, присвоить $h \leftarrow h + h/2$; затем, если $h < N$, вернуться к P3.

P5. Присвоить $m \leftarrow \lfloor m/2 \rfloor$ и, если $m \geq 1$, возвратиться к P2. ▌

Хотя смещения для сортировки генерируются не в порядке убывания, сформированная последовательность значений обеспечивает правильную работу алгоритма.

32. 4 12 11 13 2 0 8 5 10 14 1 6 3 9 16 7 15.

33. Улучшить программу можно двумя разными способами. Во-первых, полагая, что искусственный ключ K_0 равен ∞, можно опустить проверку условия $p > 0$. (Эта идея применялась, например, в алгоритме 2.2.4A.) Во-вторых, можно использовать стандартный прием оптимизации: продублировать внутренний цикл, поменяв местами присвоенные регистрам значения p и q; в результате удастся избежать присвоения $q \leftarrow p$. (Эта идея использовалась в упр. 1.1–3.)

Таким образом, предполагается, что в поле (0:3) по адресу INPUT содержится наибольшее возможное значение и нужно заменить в программе L строки, начиная с 07, следующими.

```
07  8H LD3  INPUT,2(LINK)   B'   p ← L_q. (Здесь p ≡ rI3, q ≡ rI2.)
08     CMPA INPUT,3(KEY)    B'
09     JG   4F              B'   Перейти к шагу L4 с q ↔ p, если K > K_p.
10  7H ST1  INPUT,2(LINK)   N'   L_q ← j.
11     ST3  INPUT,1(LINK)   N'   L_j ← p.
12     JMP  6F              N'   Переход на уменьшение j.
13  4H LD2  INPUT,3(LINK)   B''  p ← L_q. (Здесь p ≡ rI2, q ≡ rI3.)
14     CMPA INPUT,2(KEY)    B''
15     JG   8B              B''  Перейти к шагу L4 с q ↔ p, если K > K_p.
16  5H ST1  INPUT,3(LINK)   N''  L_q ← j.
17     ST2  INPUT,1(LINK)   N''  L_j ← p.
18  6H DEC1 1               N    j ← j − 1.
19     ENT3 0               N    q ← 0.
20     LDA  INPUT,1         N    K ← K_j.
21     J1P  4B              N    N > j ≥ 1.  ▌
```

Здесь $B'+B''=B+N-1$, $N'+N''=N-1$, так что суммарное время выполнения сортировки равно $5B+14N+N'-3$ машинных циклов. Поскольку N' равно числу элементов, справа от которых имеется нечетное число меньших элементов, величина N' имеет следующие статистические характеристики:

$$(\min 0,\ \text{ave}\ \frac{1}{2}N+\frac{1}{4}H_{\lfloor N/2\rfloor}-\frac{1}{2}H_N,\ \max N-1).$$

Трюк с присвоением значения ∞ дает аналогичное ускорение и программы S. Приведенный ниже текст программы предложен Дж. Х. Гальпериным (J. H. Halperin), который использовал эту идею и команду MOVE для сокращения времени выполнения до $(6B+11N-10)u$, полагая, что по адресу INPUT+N+1 уже находится наибольшее возможное однословное число.

```
01  START ENT2 N-1          1
02  2H    LDA  INPUT,2      N − 1
03        ENT1 INPUT,2      N − 1
04        JMP  3F           N − 1
05  4H    MOVE 1,1(1)       B
06  3H    CMPA 1,1          B + N − 1
07        JG   4B           B + N − 1
08  5H    STA  0,1          N − 1
09        DEC2 1            N − 1
10        J2P  2B           N − 1      ▌
```

Дублирование внутреннего цикла сохранит дополнительно $B/2$ или около того машинных циклов.

34. Существует $\binom{N}{n}$ последовательностей из N выборов, в которых данный список выбирается n раз; вероятность появления каждой такой последовательности равна $(1/M)^n(1-1/M)^{N-n}$, так как каждый список выбирается с вероятностью $1/M$.

35.

```
24      ENT1 0                  1        29      ENT1 0,3            N
25      ENT2 1-M                1        30      LD3  INPUT,1(LINK)  N
26   7H LD3  HEAD+M,2           M        31      J3P  *-2            N
27      J3Z  8F                 M        32   8H INC2 1              M
28      ST3  INPUT,1(LINK)   M - E       33      J2NP 7B             M  ▌
```

Замечание. Если программа M была модифицирована таким образом, чтобы отслеживать текущий конец каждого списка, то, поместив команду "ST1 END,4" между строками 19 и 20, можно сберечь время путем объединения списков так же, как и в алгоритме 5.2.5H.

36. Программа L: $A = 3$, $B = 41$, $N = 16$, время $= 496u$. Программа M: $A = 2 + 1 + 1 + 3 = 7$, $B = 2 + 0 + 3 + 3 = 8$, $N = 16$, время $= 549u$. (К приведенному значению времени выполнения программы M нужно добавить время $94u$, необходимое для выполнения дополнительной программы из упр. 35. Это позволит осуществить корректное сравнение. Операции умножения всегда замедляют выполнение программы! Обратите внимание на то, что время выполнения программы L, усовершенствованной в упр. 33, составляет только $358u$.

37. Сформулированное тождество эквивалентно тождеству

$$g_{NM}(z) = M^{-N} \sum_{n_1+\cdots+n_M=N} \left(\frac{N!}{n_1!\ldots n_M!}\right) g_{n_1}(z)\ldots g_{n_M}(z),$$

которое доказывается, как в упр. 34. Интересно, по-видимому, будет привести таблицу некоторых из этих производящих функций, чтобы выявить тенденцию, которая наблюдается при возрастании M:

$$\begin{aligned}
g_{41}(z) &= (216 + 648z + 1080z^2 + 1296z^3 + 1080z^4 + 648z^5 + 216z^6)/5184,\\
g_{42}(z) &= (945 + 1917z + 1485z^2 + 594z^3 + 135z^4 + 81z^5 + 27z^6)/5184,\\
g_{43}(z) &= (1704 + 2264z + 840z^2 + 304z^3 + 40z^4 + 24z^5 + 8z^6)/5184.
\end{aligned}$$

Если $G_M(w,z)$ — сформулированная двойная производящая функция, то дифференцирование по z дает

$$G'_M(w,z) = M\left(\sum_{n\ge 0} g_n(z)\frac{w^n}{n!}\right)^{M-1} \sum_{n\ge 0} g'_n(z)\frac{w^n}{n!}.$$

Следовательно,

$$\sum_{N\ge 0} g'_{MN}(1)\frac{M^N w^N}{N!} = Me^{(M-1)w}\left(\frac{w^2}{4}e^w\right) = \frac{M}{4}w^2 e^{Mw}.$$

Аналогично из формулы $g''_n(1) = \frac{3}{2}\binom{n}{4} + \frac{5}{3}\binom{n}{3}$ следует, что

$$\sum_{N\ge 0} g''_{MN}(1)\frac{M^N w^N}{N!} = M(M-1)e^{(M-2)w}\left(\frac{w^2}{4}e^w\right)^2 + Me^{(M-1)w}\left(\frac{3}{2}w^4 + \frac{5}{3}w^3\right)e^w.$$

Приравнивание коэффициентов при w^N дает $g'_{NM}(1) = \frac{1}{2}\binom{N}{2}M^{-1}$, $g''_{NM}(1) = \left(\frac{3}{2}\binom{N}{4} + \frac{5}{3}\binom{N}{3}\right)\times$ $\times M^{-2}$, и оказывается, что дисперсия равна $\left(\frac{1}{6}\binom{N}{3} + \frac{2M-1}{4}\binom{N}{2}\right)M^{-2}$.

38. $\sum_{j,n}\binom{N}{n}p_j^n(1-p_j)^{N-n}\binom{n}{2} = \binom{N}{2}\sum_j p_j^2$; положив $p_j = F(j/M) - F((j-1)/M)$ и $F'(x) = f(x)$, эту сумма можно привести к произведению $\binom{N}{2}/M$ и $\int_0^1 f(x)^2\,dx$, если только функция F подобрана достаточно хорошо. [Однако $\int_0^1 f(x)^2\,dx$ может оказаться слишком велико. Тонкости выбора, подходящего для *всех* ограниченных интегрируемых плотностей, приводятся в теореме 5.2.5Т. (Здесь имеется в виду интегрируемость по Риману. — *Прим. ред.*).]

39. Чтобы минимизировать величину $AC/M + BM$, нужно положить $M = \sqrt{AC/B}$, так что M — одно из целых, ближайшее (сверху или снизу) к этому значению. (В случае программы M можно было бы выбрать M пропорциональным N.)

40. Асимптотические ряды для

$$\sum_{n>N} n^{-1}(1-\alpha/N)^{n-N} = -N^{-1} + \sum_{k\ge 0}(N+k)^{-1}(1-\alpha/N)^k$$

можно получить, ограничившись значениями k до $O(N^{1+\epsilon})$, представив $(1-\alpha/N)^k$ как произведение $e^{-\alpha k/N}$ и $(1 - k\alpha^2/2N^2 + \cdots)$ и использовав формулу суммирования Эйлера; теперь выражение начинается с членов $e^\alpha E_1(\alpha)(1+\alpha^2/2N) - (1+\alpha)/2N + O(N^{-2})$. Следовательно, асимптотическое значение (15) есть $N(\ln\alpha + \gamma + E_1(\alpha))/\alpha + (1 - e^{-\alpha}(1+\alpha))/2\alpha + O(N^{-1})$. [Коэффициент при N равен ≈ 0.7966, 0.6596, 0.2880 соответственно для $\alpha = 1, 2, 10$.] Обратите внимание на то, что, как показано в упр. 5.2.2–43, $\ln\alpha + \gamma + E_1(\alpha) = \int_0^\alpha (1-e^{-t})t^{-1}\,dt$.

41. Имеем $a_k = O(\rho^k)$, поскольку из теоремы о простых числах следует, что количество простых чисел между ρ^k и ρ^{k+1} равно $\rho^{k+1}/(k+1) - \rho^k/k + O(\rho^k/k^2)$. Это значение положительно для всех достаточно больших k. Таким образом, сумма первых $\binom{k}{2}$ элементов в соотношении (10) равна $\sum_{1\le i<j\le k} b(a_i, a_j) = \sum_{1\le i<j\le k} O(\rho^{i+j})$. Отсюда получим

$$\sum_{1\le i<j\le k}\rho^{i+j} = \frac{\rho^3(\rho^k-1)(\rho^{k-1}-1)}{(\rho^2-1)(\rho-1)}.$$

(b) Если $\binom{k-1}{2} < \log_\rho N \le \binom{k}{2}$, имеем $(k-1)^2 < 2\log_\rho N$, поскольку $\rho^{2k} = O(\exp c\sqrt{\ln N})$.

Обратите внимание на то, что, поскольку $\rho \to 1$, базовая последовательность $a_1, a_2, \ldots$ становится равной последовательности простых чисел и в теореме I граница снижается до $O(N(\log N)^4(\log\log N)^{-3})$.

42. (a) [A. C. Yao, *J. Algorithms* **1** (1980), 14–50.] Можно показать, что каждая из $\binom{h}{2}$ пар списков потребует $\frac{\sqrt{\pi}}{4}g^{-2}h^{-3/2}N^{3/2} + O(N/gh)$ инверсий на каждый подмассив $(K_a, K_{a+g}, K_{a+2g}, \ldots)$, $1 \le a \le g$. Например, предположим, что $h = 12$, $g = 5$, $a = 1$, и проанализируем инверсии, когда списки $K_3 < K_{15} < K_{27} < \cdots$ и $K_7 < K_{19} < K_{31} < \cdots$ пересекают подмассив $(K_1, K_6, K_{11}, \ldots)$. После первого прохода $(K_3, K_7, K_{15}, K_{19}, K_{27}, K_{31}, \ldots)$ получается случайная 2-упорядоченная перестановка. Элементы K_j, которые нас особо заботят, имеют $j \equiv 1$ (по модулю 5) и $j \equiv 3$ или 7 (по модулю 12); следовательно, $j \equiv 51$ или 31 (по модулю 60), а нам нужно вычислить среднее значение $g(51, 31)$, где

$$g(x,y) = \sum_{j<k}\left([K_{x+ghj} > K_{y+ghk}] + [K_{y+ghj} > K_{x+ghk}]\right) + r(x,y),$$

$$r(x,y) = \sum_j [K_{\min(x,y)+ghj} > K_{\max(x,y)+ghj}] < N/gh.$$

Если $|p| \le g$ и $|q| \le g$, то получим

$$[K_{j+ph-gh} > K_{k+qh+gh}] \le [K_j > K_k] \le [K_{j+ph+gh} > K_{k+qh-gh}].$$

Значит,

$$[K_{x+ghj} > K_{y+ghk}] + [K_{y+ghj} > K_{x+ghk}]$$
$$\le [K_{x+ph+gh(j+1)} > K_{y+qh+gh(k-1)}] + [K_{y+qh+gh(j+1)} > K_{x+ph+gh(k-1)}].$$

Отсюда следует, что $g(x,y) \le g(x+ph, y+qh) + 8N/gh$. Аналогично найдем $g(x,y) \ge g(x+ph, y+qh) - 8N/gh$. Но сумма $g(x,y)$ по всем g^2 парам (x,y), таким, что $x \bmod h = b$ и $y \bmod h = c$ для любого данного $b \ne c$, и есть общее количество инверсий в случайной 2-упорядоченной перестановке $2N/h$ элементов. Таким образом, как следует из упр. 14, среднее значение $g(x,y)$ равно $g^{-2}\sqrt{\pi/128}\,(2N/h)^{3/2} + O(N/gh)$.

(b) См. C. S. Janson, D. E. Knuth, *Random Structures and Algs.* **10** (1997), 125–142. Если h и g велики, имеем $\psi(h,g) = \sqrt{\pi h/128}\,g + O(g^{-1/2}h^{1/2}) + O(gh^{-1/2})$.

43. Если $K < K_l$ после шага D3, присвоить $(K_l, \ldots, K_{j-h}, K_j) \leftarrow (K, K_l, \ldots, K_{j-h})$; в противном случае выполнять шаги D4 и D5 до тех пор, пока $K \ge K_i$. Здесь $l = 1$, если $j = h+1$, а $l \leftarrow l + 1 - h\,[l = h]$, если j увеличивается на 1. [См. H. W. Thimbleby, *Software Practice & Exper.* **19** (1989), 303–307.]

Другая идея повышения скорости выполнения программы состоит в том, чтобы при $h > 1$ выполнять сортировку только частично, не пытаясь продвинуть K_j влево далее позиции $j - h$ [см. W. Dobosiewicz, *Inf. Proc. Letters* **11** (1980), 5–6], но, кажется, этот подход требует больше смещений.

44. (a) Да. Это очевидно в случае, когда π' оказывается на один шаг "выше" π; в упр. 5.1.1–29 показано, что при таких условиях существует путь от π к любой перестановке над ней. Этот путь образован операциями транспонирования соседних элементов.

(b) Да. Аналогично, если π расположена "выше" π', то π^R расположена "ниже" π'^R.

(c) Нет; 2 1 3 не будет ни "выше", ни "ниже" 3 1 2, но $2\,1\,3 \le 3\,1\,2$.

[Частичное упорядочение $\pi \le \pi'$ было впервые проанализировано Ч. Эресманом в контексте алгебраической топологии (C. Ehresmann, *Annals of Math.* (2) **35** (1934), 396–443, §20). Многие математики называют его теперь порядком Брюа перестановок.]

РАЗДЕЛ 5.2.2

1. Нет, в ней на $2m+1$ инверсий *меньше*, где $m \ge 0$ — число элементов a_k, таких, что $i < k < j$ и $a_i > a_k > a_j$. (Следовательно, любая обменная сортировка, в конце концов, приводит к упорядоченной перестановке.)

2. (a) 6. (b) [A. Cayley, *Philos. Mag.* **34** (1849), 527–529.] Рассмотрим циклическое представление перестановки π. Если поменять местами элементы *одного и того же* цикла, то число циклов *увеличится* на 1; если поменять местами элементы *разных* циклов, то число циклов *уменьшится* на 1. (Это, по существу, содержание упр. 2.2.4–3.) Полностью рассортированная перестановка характеризуется тем, что она имеет n циклов. Следовательно, $\mathrm{xch}(\pi)$ равно n минус число циклов перестановки π. [Алгоритм 5.2.3S выполняет ровно $\mathrm{xch}(\pi)$ операций обмена записей; см. упр. 5.2.3–4.)

3. Да; относительное расположение равных элементов никогда не меняется.

4. Это вероятность того, что в таблице инверсий будет $b_1 > \max(b_2, \ldots, b_n)$. Вероятность равна

$$\Bigl(\sum_{1\le k<n} k!\,k^{n-k-1}\Bigr)\Bigg/ n! = \sqrt{\pi/2n} + O(n^{-1}) = \text{ пренебрежимо малая величина.}$$

5. Можно считать, что $r > 0$. Пусть $b'_i = (b_i - r + 1)[b_i \ge r]$ — таблица инверсий после $r-1$ проходов. Если $b'_i > 0$, то элементу i предшествует b'_i больших элементов, наибольший из которых всплывает, по крайней мере, до позиции $b'_i + i$ (так как имеется i элементов $\le i$).

Кроме того, если элемент j — крайний справа среди всех элементов, которые предстоит обменять, то $b'_j > 0$ и после r-го прохода $\texttt{BOUND} = b'_j + j - 1$.

6. *Решение 1.* Элемент, наиболее удаленный вправо от своего конечного положения, перемещается на один шаг влево при каждом просмотре, кроме последнего. *Решение 2* (более высокого уровня). Из упр. 5.1.1-8, ответ (f), $a'_i - i = b_i - c_i$ при $1 \le i \le n$, где $c_1 c_2 \ldots c_n$ — двойственная таблица инверсий. Если $b_j = \max(b_1, \ldots, b_n)$, то $c_j = 0$.

7. $\left(2(n+1)(1+P(n)-P(n+1)) - P(n) - P(n)^2\right)^{1/2} = \sqrt{(2-\pi/2)n} + O(1)$.

8. При $i < k+2$ имеется $j+k-i+1$ способов выбора b_i; при $k+2 \le i < n-j+2$ имеется $j-1$ /'-1 способов; при $i \ge n-j+2$ имеется $n-i+1$ способов.

10. (a) Если $i = 2k-1$, то из $(k-1,\ a_i - k)$ в $(k,\ a_i - k)$. Если $i = 2k$, то из $(a_i - k,\ k-1)$ в $(a_i - k,\ k)$. (b) Шаг a_{2k-1} выше диагонали $\iff k \le a_{2k-1} - k \iff a_{2k-1} \ge 2k \iff a_{2k-1} > a_{2k} \iff a_{2k} \le 2k-1 \iff a_{2k} - k \le k-1 \iff$ шаг a_{2k} выше диагонали. Если поменять эти элементы местами, то поменяются местами горизонтальный и вертикальный шаги. (c) Шаг a_{2k+d} будет, по крайней мере, на m единиц ниже диагонали $\iff k+m-1 \ge a_{2k+d} - (k+m) + m \iff a_{2k+d} < 2k+m \iff a_{2k} \ge 2k+m \iff a_{2k} - k \ge k+m \iff$ шаг a_{2k}. (Если $a_{2k+d} < 2k+m$ и $a_{2k} < 2k+m$, то имеется не менее $(k+m)+k$ элементов, меньших, чем $2k+m$, а это невозможно. Если $a_{2k+d} \ge 2k+m$ и $a_{2k} \ge 2k+m$, то один из знаков "$\ge$" должен быть знаком "$>$"; но невозможно поместить все элементы $\le 2k+m$ в менее чем $(k+m)+k$ позиций. Следовательно, $a_{2k+2m-1} < a_{2k}$ тогда и только тогда, когда $a_{2k+2m-1} < 2k+m$, т. е. когда $2k+m \le a_{2k}$. Довольно неожиданный результат!)

11. 16 10 13 5 14 6 9 2 15 8 11 3 12 4 7 1 (61 обмен). Ответ получается в результате анализа решеточной диаграммы. Ситуация усложняется, если N велико; в общем случае множество $\{K_2, K_4, \ldots\}$ должно быть таким: $\{1, 2, \ldots, M-1, M, M+2, M+4, \ldots, 2\lfloor N/2 \rfloor - M\}$; и его перестановка должна максимизировать число обменов для $\lfloor N/2 \rfloor$ элементов. Здесь $M = \lceil 2^k/3 \rceil$, где k максимизирует $k\lfloor N/2 \rfloor - \frac{1}{9}\left((3k-2)2^{k-1} + (-1)^k\right)$. Суммарное максимальное число операций обмена записей равно произведению $1 - 2\lg\lg N/\lg N + O(1/\log N)$ и числа сравнений [R. Sedgewick, *SICOMP* **7** (1978), 239–272].

12. В следующей программе, написанной В. Панни (W. Panny), команда AND не используется — для этого шаг M4 выполняется при $i = r + 2kp + s$, $k \ge 0$ и $0 \le s < p$. Здесь $\texttt{TT} \equiv 2^{t-1}$, $p \equiv$ rI1, $r \equiv$ rI2, $i \equiv$ rI3, $i + d - N \equiv$ rI4 и $p - 1 - s \equiv$ rI5; полагается, что $N \ge 2$.

```
01  START ENT1 TT              1      M1. Инициализация p. p ← 2^(t-1).
02  2H    ENT2 TT              T      M2. Инициализация q, r, d.
03        ST2  Q(1:2)          T      q ← 2^(t-1).
04        ENT2 0               T      r ← 0.
05        ENT4 0,1             T      rI4 ← d.
05  3H    ENT3 0,2             A      M3. Цикл по i. i ← r.
07        INC4 -N,3            A      rI4 ← i + d − N.
08  8H    ENT5 -1,1          D + E    s ← 0.
09  4H    LDA  INPUT+1,3       C      M4. Сравнение/обмен R_(i+1) : R_(i+d+1).
10        CMPA INPUT+N+1,4     C
11        JLE  *+4             C      Переход, если K_(i+1) ≤ K_(i+d+1).
12        LDX  INPUT+N+1,4     B
13        STX  INPUT+1,3       B      R_(i+1) ↔ R_(i+d+1).
14        STA  INPUT+N+1,4     B
15        J5Z  7F              C      Переход, если s = p − 1.
16        DEC5 1             C − D    s ← s + 1.
17        INC3 1             C − D    i ← i + 1.
18        INC4 1             C − D
```

19		J4N	4B	$C-D$	Повторить цикл, если $i+d<N$.
20		JMP	5F	E	Иначе перейти к M5.
21	7H	INC3	1,1	D	$i \leftarrow i+p+1$.
22		INC4	1,1	D	
23		J4N	4B	D	Повторить цикл, если $i+d<N$.
24	5H	ENT2	0,1	A	*M5. Цикл по q.* $r \leftarrow p$.
25	Q	ENT4	*	A	rI4 $\leftarrow q$.
26		ENTA	0,4	A	
27		SRB	1	A	
28		STA	Q(1:2)	A	$q \leftarrow q/2$.
29		DEC4	0,1	A	rI4 $\leftarrow d$.
30		J4P	3B	A	Перейти к шагу M3, если $d \neq 0$.
31	6H	ENTA	0,1	T	*M6. Цикл по p.*
32		SRB	1	T	
33		STA	*+1(1:2)	T	
34		ENT1	*	T	$p \leftarrow \lfloor p/2 \rfloor$.
35		J1NZ	2B	T	Перейти к шагу M2, если $p \neq 0$. ▌

Время выполнения зависит от шести параметров, из которых лишь один зависит от исходных данных (остальные пять являются функциями только от N): $T=t$, числа "внешних циклов"; $A=t(t+1)/2$, числа просмотров, или "внутренних циклов"; $B=$ числу обменов (переменному); $C=$ числу сравнений; $D=$ числу блоков последовательных сравнений; $E=$ числу неполных блоков. Если $N=2^t$, то можно показать, что $D=(t-2)N+t+2$ и $E=0$. Для данных из табл. 1 $T=4$, $A=10$, $B=3+0+1+4+0+0+8+0+4+5=25$, $C=63$, $D=38$, $E=0$, так что общее время выполнения равно $11A+6B+10C+2E+12T+1=939u$.

Панни показал, что в общем случае $D=e_1(N+1)-2(2^{e_1}-1)$, $E=\binom{e_1-e_r}{2}+(e_1+e_2+\cdots+e_{r-1})-(e_1-1)(r-1)$, если $N=2^{e_1}+\cdots+2^{e_r}$.

13. Нет, так же, как и алгоритмы Q и R.

14. (a) Если $p=1$, то при последнем слиянии выполняем $(2^{t-1}-0)+(2^{t-1}-1)+(2^{t-1}-2)+(2^{t-1}-4)+\cdots+(2^{t-1}-2^{t-2})=(t-1)2^{t-1}+1$ сравнений; (b) $x_t=x_{t-1}+\frac{1}{2}(t-1)+2^{-t}=\cdots=x_0+\sum_{0\le k<t}(\frac{1}{2}k+2^{-k-1})=\frac{1}{2}\binom{t}{2}+1-2^{-t}$. Следовательно, $c(2^t)=2^{t-2}(t^2-t+4)-1$.

15. (a) Рассмотрите число сравнений, таких, что $i+d=N$; затем примените индукцию по r. (b) Если $b(n)=c(n+1)$, то имеем $b(2n)=a(1)+\cdots+a(2n)=a(0)+a(1)+a(1)+\cdots+a(n-1)+a(n)+x(1)+x(2)+\cdots+x(2n)=2b(n)+y(2n)-a(n)$; аналогично $b(2n+1)=2b(n)+y(2n+1)$. (c) См. упр. 1.2.4–42. (d) Весьма трудоемкие вычисления выражения $(z(N)+2z(\lfloor N/2\rfloor)+\cdots)-a(N)$ с использованием таких формул, как

$$\sum_{k=0}^{n}2^k(n-k)=2^{n+1}-n-2,\qquad \sum_{k=0}^{n}2^k\binom{n-k}{2}=2^{n+1}-\binom{n+2}{2}-1,$$

приводят к результату

$$c(N)=N\Big(\frac{1}{2}\binom{e_1}{2}+2e_1-1\Big)-2^{e_1}(e_1-1)-1$$
$$+\sum_{j=1}^{r}2^{e_j}\Big(e_1+\cdots+e_{j-1}-j(e_1-1)+\frac{1}{2}\binom{e_1-e_j}{2}\Big).$$

16. Рассмотрим $\binom{2n}{n}$ путей на решетке из $(0,0)$ в (n,n), как на рис. 11 и 18, и присвоим серии от (i,j) до $(i+1,j)$ вес $f(i-j)$, если $i\ge j$, и $f(j-i-1)+1$, если $i<j$; здесь $f(k)$ — число изменений разрядов $b_r\neq b_{r+1}$ в двоичном представлении $k=(\ldots b_2b_1b_0)_2$. Если $N=2n$, то общее число обменов в окончательном слиянии равно $\sum_{0\le j\le i<n}(2f(j)+$

$1)\binom{2i-j}{i-j}\binom{2n-2i+j-1}{n-i-1}$. Р. Седгевик показал, что в общем случае f эта сумма упрощается и приводится к виду $\frac{n}{2}\binom{2n}{n}+2\sum_{k\ge1}\binom{2n}{n-k}\sum_{0\le j<k}f(j)$; затем он использовал метод гамма-функции и получил асимптотическое выражение

$$\binom{2n}{n}\left(\frac{1}{4}n\lg n+\left(\lg\frac{\Gamma(1/4)^2}{2\pi}+\frac{1}{4}+\frac{\gamma+2}{4\ln 2}+\delta(n)\right)n+O(\sqrt{n}\log n)\right),$$

где $\delta(n)$ — периодическая функция $\lg n$ с ограниченной амплитудой .0005. Следовательно, в среднем около 1/4 сравнений приведет к обмену при $n\to\infty$. [*SICOMP* **7** (1978), 239–272; см. также Flajolet, Odlyzko, *SIAM J. Discrete Math.* **3** (1990), 238–239.]

17. Значение K_{N+1} проверяется при сортировке подмассива, для которого $r=N$ и K_l — наибольший ключ. K_0 проверяется на шаге Q9, когда минимум слева направо погружается до позиции R_1.

18. На шагах Q3 и Q4 до перехода к Q5 выполняется единственная замена i на j; процесс разделения подмассива $R_l\ldots R_r$ заканчивается при $j=\lceil(l+r)/2\rceil$ на шаге Q7, т. е. подмассив разделяется по возможности точно пополам. (Количественно это выражается в том, что формулы (17) заменяются формулами $A=1$, $B=\lfloor(N-1)/2\rfloor$, $C=N+(N \bmod 2)$; это, по существу, *наилучший* случай для алгоритма (см. также упр. 27, приведенное ниже), за исключением того, что $B\approx\frac{1}{2}C$. Если на шагах Q3 и Q4 заменить знаки "<" знаками "$\le$", то алгоритм вообще не будет сортировать данные. Даже если предполагать, что в (13) стоят знаки "<", все равно наш алгоритм поменяет местами R_0 с R_1, затем в третьей фазе разделения переместит исходную запись R_0 в позицию R_2 и т. д. Настоящая катастрофа!

19. Да, подмассивы можно обрабатывать в любом порядке. Но в очереди будет $\Omega(N/\sqrt{\log N})$ элементов, если каждая очередная итерация будет разбивать массив ровно пополам, в то время как стек гарантированно остается меньшим по объему (см. следующее упражнение).

20. $\max(0,\lfloor\lg(N+2)/(M+2)\rfloor)$. (Самый плохой случай — это когда $N=2^k(M+2)-1$ и все подмассивы разделяются точно пополам.)

21. На шаге Q6 точно t записей перемещается в область $R_{s+1}\ldots R_N$, поскольку $B=t$. Когда фаза разбиения завершается, то $j=s$. Следовательно, $C-C'=N+1-s$ есть количество вычитаний 1 из j. На шаге Q7 мы должны получить $i=s+1$, если ключи различны, поскольку $i=j$ влечет за собой $K_j=K$. Таким образом, $C'=s$.

22. Указанное соотношение для $A_N(z)$ легко вывести, поскольку $A_{s-1}(z)A_{N-s}(z)$ — производящая функция для величины A после независимой сортировки случайных последовательностей длиной $s-1$ и $N-s$. Аналогичным образом получаем соотношения

$$B_N(z)=\sum_{s=1}^{N}\sum_{t=0}^{s}b_{stN}z^tB_{s-1}(z)B_{N-s}(z),$$

$$C_N(z)=\frac{1}{N}\sum_{s=1}^{N}z^{N+1}C_{s-1}(z)C_{N-s}(z),$$

$$D_N(z)=\frac{1}{N}\sum_{s=1}^{N}D_{s-1}(z)D_{N-s}(z),$$

$$E_N(z)=\frac{1}{N}\sum_{s=1}^{N}E_{s-1}(z)E_{N-s}(z),$$

$$S_N(z) = \frac{1}{N}\sum_{s=1}^{N} z^{[M+1<s<N-M]} S_{s-1}(z) S_{N-s}(z)$$

при $N > M$. Здесь b_{stN} — вероятность того, что s и t примут данные значения в последовательности длиной N, а именно

$$\binom{s-1}{t}\binom{N-s}{t}\Big/ N\binom{N-1}{s-1}.$$

Последнее выражение равно произведению трех сомножителей: числа перестановок множества $\{1,\ldots,s-1\}$, числа перестановок множества $\{s+1,\ldots,N\}$ и числа способов взаимной замены t элементов из одного подмассива и t элементов из другого подмассива. Первый сомножитель есть произведение $(1/N!)$ и $(s-1)!$, второй сомножитель равен $(N-s)!$, а третий равен $\binom{s-1}{t}\binom{N-s}{t}$. При $0 \le N \le M$ получим $B_N(z) = C_N(z) = S_N(z) = 1$, $D_N(z) = \prod_{k=1}^{N}((1+(k-1)z)/k)$ и $E_N(z) = \prod_{k=1}^{N}((1+z+\cdots+z^{k-1})/k)$.

[Интересно проанализировать поведение этой производящей функции при большом N; последовательность, аналогичная $C_N(z)$, но с z^{N+1}, замененным на z^{N-1}, как известно, сходится к распределению, отличному от нормального, что не позволяет проанализировать его полностью. (См. статью P. Hennequin, M. Regnier, U. Rösler, *RAIRO Theoretical Informatics and Applications* **23** (1989), 317–333; **23** (1989), 335–343; **25** (1991), 85–100.)]

23. При $N > M$ получаем $A_N = 1 + (2/N)\sum_{0\le k<N} A_k$; $B_N = \sum_{0\le t<s\le N} b_{stN}(t + B_{s-1} + B_{N-s}) = (1/N)\sum_{s=1}^{N}((s-1)(N-s)/(N-1)+B_{s-1}+B_{N-s}) = (N-2)/6+(2/N)\sum_{0\le k<N} B_k$ [см. упр. 22]; $D_N = (2/N)\sum_{0\le k<N} D_k$; E_N получается аналогично. При $N > 2M+1$ получаем $S_N = (2/N)\sum_{0\le k<N} S_k + (N-2M-2)/N$. Каждое из этих рекуррентных соотношений имеет форму (20) при некоторой функции f_n.

24. Рекуррентное соотношение $C_N = N-1+(2/N)\sum_{0\le k<N} C_k$ при $N > M$ имеет решение $(N+1)(2H_{N+1}-2H_{M+2}+1-4/(M+2)+2/(N+1))$. (Таким образом, мы могли бы сэкономить около $4N/M$ сравнений. Но для каждого сравнения потребуется больше времени, если за ним последует анализ i и j. В результате мы проиграем, если только потери при сравнении двух ключей не превысят в $\frac{1}{2}M\ln N$ раз потери от сравнения двух регистров. Многие теоретические выводы относительно сортировки не удалось реализовать на практике, поскольку предлагаемые "усовершенствования" превращали быструю сортировку в существенно менее быструю!)

25. (Многократно примените формулы (17), подставив $s = 1$.) $A = N-M$, $B = 0$, $C = \binom{N+2}{2} - \binom{M+2}{2}$, $D = E = S = 0$.

26. Действительно, нет ничего хуже, чем сортировать

$$1\ 2\ 3\ \ldots\ N{-}M\ N\ N{-}1\ \ldots\ N{-}M{+}1.$$

Часть этой последовательности $N\ M{-}1\ M{-}2\ \ldots\ 1\ M\ M{+}1\ \ldots\ N{-}1$ почти так же плоха. Но это лишь чуть-чуть хуже, чем в упр. 25, поскольку при таком наборе получим $D = M-1$, $E = \binom{M}{2}$.

27. 12 2 3 1 8 6 7 5 9 10 11 4 16 14 15 13 20 18 19 17 21 22 23; требует $546u$. Можно показать, что наилучшим при $N = 3(M+1)2^k - 1$ будет случай, когда подмассивы при каждом разбиении делятся пополам до тех пор, пока не достигнут размера $3M+2$. Затем нужно использовать деление на треть, чтобы избежать переполнения стека. Получим $A = 3\cdot 2^k - 1$, $C = (k+\frac{5}{3})(N+1)$, $S = 2^k - 1$, $B = D = E = 0$. (Найти лучший случай для любого M и N — задача очень интересная, но и очень сложная.)

28. Рекуррентное соотношение

$$C_n = n+1+\frac{2}{\binom{n}{3}}\sum_{k=1}^{n}(k-1)(n-k)C_{k-1}$$

можно преобразовать в

$$\binom{n}{3}C_n - 2\binom{n-1}{3}C_{n-1} + \binom{n-2}{3}C_{n-2} = 2(n-1)(n-2)+2(n-2)C_{n-2}.$$

29. В общем случае рассмотрим рекуррентное соотношение

$$C_n = n+1+\frac{2}{\binom{n}{2t+1}}\sum_{k=1}^{n}\binom{k-1}{t}\binom{n-k}{t}C_{k-1},$$

которое возникает, когда разделение управляется медианой $2t+1$ элементов. Полагая $C(z)=\sum_n C_n z^n$, его можно преобразовать в $(1-z)^{t+1}C^{(2t+1)}(z)/(2t+2)! = 1/(1-z)^{t+2} + C^{(t)}(z)/(t+1)!$. Пусть $f(x)=C^{(t)}(1-x)$; тогда $p_t(\vartheta)f(x)=(2t+2)!/x^{t+2}$, где ϑ — оператор $x(d/dx)$ и $p_t(x)=(t-x)^{\underline{t+1}} - (2t+2)^{\underline{t+1}}$. Общее решение уравнения: $(\vartheta-\alpha)g(x)=x^\beta$ имеет вид $g(x)=x^\beta/(\beta-\alpha)+Cx^\alpha$ при $\alpha\neq\beta$ и $g(x)=x^\beta(\ln x + C)$ при $\alpha=\beta$. Имеем $p_t(-t-2)=0$, так что общее решение этого дифференциального уравнения таково:

$$C^{(t)}(z) = (2t+2)!\ln(1-z)/p_t'(-t-2)(1-z)^{t+2} + \sum_{j=0}^{t} c_j(1-z)^{\alpha_j},$$

где $\alpha_0,\ldots,\alpha_t$ — корни уравнения $p_t(x)=0$, а константы c_i зависят от начальных значений $C_t,\ldots,C_{2t}$. Удобное тождество

$$\frac{1}{(1-z)^{m+1}}\ln\left(\frac{1}{1-z}\right) = \sum_{n\geq 0}(H_{n+m}-H_m)\binom{n+m}{m}z^n, \qquad m\geq 0,$$

приводит к удивительно простому решению в *замкнутом виде*

$$C_n = \frac{H_{n+1}-H_{t+1}}{H_{2t+2}-H_{t+1}}(n+1) + \frac{1}{n!}\sum_{j=0}^{t} c_j(-\alpha_j)^{\overline{n-t}},$$

из которого легко выводится асимптотическая формула. (Главный член $n\ln n/\ (H_{2t+2}-H_{t+1})$ выделил М. ван Эмден [см. M. H. van Emden, *CACM* **13** (1970), 563–567], используя теоретико-информационный подход. Действительно, предположим, что необходимо проанализировать произвольный процесс разделения, такой, что в левом подмассиве будет содержаться xN элементов с асимптотической вероятностью $\int_0^x f(x)\,dx$ при $N\to\infty$, для $0\leq x\leq 1$. Ван Эмден доказал, что среднее число сравнений, необходимых для полной сортировки всего массива, имеет асимптотическое выражение $\alpha^{-1}n\ln n$, где $\alpha=-1/\int_0^1(f(x)+f(1-x))x\ln x\,dx$. Эта формула применима к обменной поразрядной сортировке, а также к быстрой сортировке и другим методам. См. также H. Hurwitz, *CACM* **14** (1971), 99–102.)

30. *Решение 1* (представляет, скорее всего, историческую ценность). Каждый подмассив можно описывать четырьмя величинами (l,r,k,X), где l и r — границы (как и ранее), k указывает число слов в ключах, о которых известно, что они равны во всем подмассиве, а X — нижняя граница $(k+1)$-х слов в ключах. В предположении, что ключи неотрицательны, имеем вначале $(l,r,k,X)=(1,N,0,0)$. При разделении массива полагаем K равным $(k+1)$-му слову проверяемого ключа K_q. Если $K>X$, то при разделении все ключи $\geq K$ оказываются справа, а все ключи $<K$ — слева (если смотреть каждый раз только на $(k+1)$-е слово ключа). Соответственно разделенные подмассивы получают описания

$(l, j-1, k, X)$ и (j, r, k, K). Но если $K = X$, то при разделении все ключи $> K$ попадают вправо, а все ключи $\leq K$ [фактически $= K$] — влево; разделенные подмассивы получают описания соответственно $(l, j, k+1, 0)$ и $(j+1, r, k, K)$. В обоих случаях нельзя быть уверенным в том, что запись R_j заняла окончательное положение, поскольку мы не просмотрели $(k+2)$-е слова. Для правильной работы с граничными условиями необходимо внести другие очевидные изменения. Добавив в описание пятый компонент, "верхняя граница", можно сделать этот метод симметричным относительно левой и правой частей массива.

Решение 2 [Bentley, Sedgewick, *SODA* **8** (1997), 360–369]. Пусть, как и в решении 1, K будет $(k+1)$-ым словом в K_q в подмассиве, описанном (l, r, k). Но далее используйте алгоритм из упр. 41 для разделения подмассива на три части, $(l, i-1, k)$, $(i, j, k+1)$, $(j+1, r, k)$, для случаев $<K$, $=K$, $>K$. Этот подход, который авторы назвали *multikey quicksort* (быстрая сортировка по мультиключу), дает значительно лучшие результаты, чем решение 1, и является вполне достойным конкурентом самых быстрых методов сортировки символьных строк.

31. Выполните обычное разделение, и пусть R_1 попадет в позицию R_s. Если $s = m$, то на этом можно закончить. Если $s < m$, воспользуйтесь тем же методом для нахождения $(m-s)$-го наименьшего элемента в правом подмассиве. Если же $s > m$, найдите m-й наименьший элемент в левом подмассиве. [*CACM* **4** (1961), 321–322; **14** (1971), 39–45.]

В работе R. G. Dromey, *Software Practice & Experience* **16** (1986), 981–986, показано, что понадобится меньше сравнений и перезаписей, если прекращать каждую фазу разделения, как только i или j достигнет позиции m.

32. Рекуррентное соотношение здесь следующее: $C_{11} = 0$ и $C_{nm} = n + 1 + (A_{nm} + B_{nm})/n$ при $n > 1$, где

$$A_{nm} = \sum_{1 \leq s < m} C_{(n-s)(m-s)} \qquad \text{и} \qquad B_{nm} = \sum_{m < s \leq n} C_{(s-1)m}$$

при $1 \leq m \leq n$. Поскольку $A_{(n+1)(m+1)} = A_{nm} + C_{nm}$ и $B_{(n+1)m} = B_{nm} + C_{nm}$, можно сначала найти формулу для $D_n = (n+1)C_{(n+1)(m+1)} - nC_{nm}$, а затем сложить эти формулы и получить ответ $2((n+1)H_n - (n+2-m)H_{n+1-m} - (m+1)H_m + n + \frac{5}{3}) - \frac{1}{3}\delta_{mn} - \frac{1}{3}\delta_{m1} - \frac{2}{3}\delta_{mn}\delta_{m1}$. При $n = 2m-1$ он примет вид $4m(H_{2m-1} - H_m) + 4m - 4H_m + \frac{4}{3}(1 - \delta_{m1}) = (4 + 4\ln 2)m - 4\ln m - 4\gamma - \frac{5}{3} + O(m^{-1}) \approx 3.39n$. [См. D. E. Knuth, *Proc. IFIP Congress* (1971), 19–27.]

Другое решение является следствием теории, изложенной в разделе 6.2.2. Предположим, что ключи — это $\{1, 2, \ldots, n\}$, и пусть X_{jk} — число общих предков узлов j и k в дереве двоичного поиска, соответствующем быстрой сортировке. Тогда число сравнений, которые необходимо выполнить для реализации алгоритма из упр. 31, можно представить в виде $\sum_{j=1}^{n} X_{jm} + X_{mm} - 2$[узел m есть лист]. Вероятность того, что узел i является общим предком узлов j и k в случайном дереве двоичного поиска, есть $1/(\max(i, j, k) - \min(i, j, k) + 1)$. Среднее число сравнений получим, исходя из того, что $\mathrm{E}\, X_{jk} = H_j + H_{n+1-k} + 1 - 2H_{k-j+1}$ при $1 \leq j \leq k$ и $\Pr$(узел m есть лист) $= \Pr$(за m не следует $m \pm 1$ в случайной перестановке) $= \frac{1}{3} + \frac{1}{6}\delta_{m1} + \frac{1}{6}\delta_{mn} + \frac{1}{3}\delta_{m1}\delta_{mn}$. [См. R. Raman, *SIGACT News* **25**, 2 (June, 1994), 86–89.]

Анализ аналогичного алгоритма, в котором используется разбиение по методу медианы трех значений, приведен в работе Kirschenhofer, Prodinger, Martínez, *Random Structures и Algorithms* **10** (1997), 143–156. Асимптотически более быстрый метод анализируется в упр. 5.3.3–24.

33. Действуйте, как на первой итерации поразрядной обменной сортировки, но вместо первого разряда числа используйте знаковый разряд.

34. В каждой итерации после обнаружения, по крайней мере, одного разряда, имеющего значение 0, и одного разряда, имеющего значение 1, т. е. после первого же обмена, можно не

проверять условие $i \le j$. Таким образом можно уменьшить время выполнения программы R примерно на $2C$ машинных циклов.

35. $A = N-1$, $B = (\min 0,\ \text{ave}\ \frac{1}{4}N\lg N,\ \max \frac{1}{2}N\lg N)$, $C = N\lg N$, $G = \frac{1}{2}N$, $K = L = R = 0$, $S = \frac{1}{2}N - 1$, $X = (\min 0,\ \text{ave}\ \frac{1}{2}(N-1),\ \max N-1)$. В общем случае параметры A, C, G, K, L, R и S зависят от значений ключей в последовательности, но не от их начального расположения; начальное расположение ключей влияет только на параметры B и X.

36. (a) $\sum \binom{n}{k}\binom{k}{j}(-1)^{k+j}a_j = \sum \binom{n}{j}\binom{n-j}{k-j}(-1)^{k-j}a_j = \sum \binom{n}{j}\delta_{nj}a_j = a_n$. (b) $\langle\delta_{n0}\rangle$; $\langle-\delta_{n1}\rangle$; $\langle(-1)^m\delta_{nm}\rangle$; $\langle(1-a)^n\rangle$; $\langle\binom{n}{m}(-a)^m(1-a)^{n-m}\rangle$. (c) Запишем соотношение, которое требуется доказать, в виде $x_n = y_n = a_n + z_n$. Тогда из п. (a) получим $y_n = a_n + z_n$. Также $2^{1-n}\sum_{k\ge 2}\binom{n}{k}y_k = z_n$; следовательно, y_n удовлетворяет тому же рекуррентному соотношению, что и x_n. [Некоторое обобщение этого результата приводится в упр. 53 и 6.3–17. Оказывается, *непосредственно* доказать, что $\hat{x}_n = \hat{a}_n 2^{n-1}/(2^{n-1}-1)$, отнюдь непросто!]

37. $\langle\sum_m c_m\binom{n}{2m}2^{-n}\rangle$ при произвольной последовательности констант c_0, c_1, c_2, ... [Из этого ответа, хотя он и верен, не сразу видно, что $\langle 1/(n+1)\rangle$ и $\langle n-\delta_{n1}\rangle$ принадлежат к числу таких последовательностей! Последовательности вида $\langle a_n + \hat{a}_n\rangle$ всегда являются двойственными по отношению к самим себе. Заметим, что в терминах производящих функций $A(z) = \sum a_n z^n/n!$ имеем $\hat{A}(z) = e^z A(-z)$; следовательно, равенство $A = \hat{A}$ эквивалентно утверждению о том, что $A(z)e^{-z/2}$ — четная функция.]

38. Итерация разделения, порождающая левый подмассив длиной s и правый подмассив длиной $N-s$, дает следующие вклады в суммарное время выполнения:

$$A = 1, \quad B = t, \quad C = N, \quad K = \delta_{s1}, \quad L = \delta_{s0}, \quad R = \delta_{sN}, \quad X = h,$$

где t — число таких ключей среди $K_1, \ldots, K_s$, разряд b которых равен 1, а h — это разряд b ключа K_{s+1}; если $s = N$, то $h = 0$ (см. (17)). Это приводит к таким рекуррентным уравнениям, как

$$\begin{aligned} B_N &= 2^{-N}\sum_{0\le t\le s\le N}\binom{s}{t}\binom{N-s}{t}(t + B_s + B_{N-s}) \\ &= \frac{1}{4}(N-1) + 2^{1-N}\sum_{s\ge 2}\binom{N}{s}B_s, \quad \text{где } N \ge 2; \qquad B_0 = B_1 = 0 \end{aligned}$$

(см. упр. 23). Применив для решения этих уравнений метод, описанный в упр. 36, получим формулы $A_N = V_N - U_N + 1$, $B_N = \frac{1}{4}(U_N + N - 1)$, $C_N = V_N + N$, $K_N = N/2$, $L_N = R_N = \frac{1}{2}(V_N - U_N - N) + 1$, $X_N = \frac{1}{2}(A_N - L_N)$. Ясно, что $G_N = 0$.

39. После каждой итерации быстрой сортировки, по крайней мере, один элемент попадает в свою окончательную позицию, но этого не происходит при обменной поразрядной сортировке (см. табл. 3).

40. Если переключаться на метод простых вставок всякий раз, как только на шаге R2 будет $r - l < M$, то проблема разрешится, если только не встретится более M равных элементов. Если последнее весьма вероятно, то всегда, когда на шаге R8 будет $j < l$ или $j = r$, можно проверить условие $K_l = \cdots = K_r$.

41. В работе Lutz M. Wegner, *IEEE Trans.* **C-34** (1985), 362–367, проанализировано несколько подходов, из которых описанный ниже нам представляется наиболее подходящим для использования на практике. Он несколько упрощен в работе Bentley, McIlroy, *Software Practice & Exp.* **23** (1993), 1256–1258. Основная идея состоит в том, чтобы работать с массивом, разделенным на пять частей

$= K$	$< K$	?	$> K$	$= K$

$l \qquad a \qquad b \qquad c \qquad d \qquad r$

до тех пор, пока средняя часть не станет пустой. Затем нужно перебросить две крайние части в середину.

D1. [Начальная установка.] Установить $a \leftarrow b \leftarrow l$, $c \leftarrow d \leftarrow r$.

D2. [Увеличивать b до тех пор, пока не станет $K_b > K$.] Если $b \le c$ и $K_b < K$, увеличить b на 1 и повторить этот шаг. Если $b \le c$ и $K_b = K$, выполнить обмен $R_a \leftrightarrow R_b$, увеличить a и b на 1 и повторить этот шаг.

D3. [Уменьшать c до тех пор, пока не станет $K_c < K$.] Если $b \le c$ и $K_c > K$, уменьшить c на 1 и повторить этот шаг. Если $b \le c$ и $K_c = K$, выполнить обмен $R_c \leftrightarrow R_d$, уменьшить c и d на 1 и повторить этот шаг.

D4. [Замена.] Если $b < c$, выполнить обмен $R_b \leftrightarrow R_c$, увеличить b на 1, уменьшить c на 1 и вернуться к шагу D2.

D5. [Очистка.] Выполнить обмен $R_{l+k} \leftrightarrow R_{c-k}$ при $0 \le k < \min(a-l, b-a)$; также выполнить обмен $R_{b+k} \leftrightarrow R_{r-k}$ при $0 \le k < \min(d-c, r-d)$. Окончательно установить $i \leftarrow l+b-a$, $j \leftarrow r-d+c$. ▌

Совершенно очевидная модификация на шаге D1 должна обеспечить эффективную обработку вырожденных случаев и обеспечить $a < b$ и $c < d$ до перехода к D2. Тогда исчезнет необходимость в проверках "$b \le c$" на шагах D2 и D3 (см. упр. 24). Более того, подобная замена позволит избежать обмена записи с самой собой.

Одно из приложений сортировки — сбор записей с равными ключами вместе. Таким образом, разделение на три части зачастую более предпочтительно, чем разделение на две части, предусматриваемое алгоритмом Q. Операции обмена на шаге D5 выполняются эффективно, поскольку все записи с ключами, равными K, к этому времени уже заняли свои окончательные позиции.

Это упражнение придумано В. Г. И. Фейеном (W. H. J. Feijen), который назвал его "проблемой голландского флага": "Задано множество объектов красного, белого и синего цветов, случайным образом разбросанных между тремя колонками. Нужно придумать способ взаимного обмена пар объектов таким образом, чтобы красные собрались в верхней части колонки, а синие — внизу, причем каждый объект должен просматриваться только один раз и разрешается использовать минимальное число дополнительных переменных для управления процессом". [См. E. W. Dijkstra, *A Discipline of Programming* (Prentice-Hall, 1976), Chapter 11*.]

42. Это особый случай теоремы Карпа (см. R. M. Karp, *JACM* **41** (1994), 1136–1150, §2.8). Существенно более строгая асимптотическая граница для хвоста распределения при быстрой сортировке была получена в работе C. J. H. McDiarmid, R. B. Hayward, *J. Algorithms* **21** (1996), 476–507.

43. При $a \to 0+$, как следует из упр. 1.2.7–24, имеем

$$\int_0^1 y^{a-1}(e^{-y}-1)\,dy + \int_1^\infty y^{a-1}e^{-y}\,dy = \Gamma(a) - 1/a = (\Gamma(a+1)-\Gamma(1))/a \to \Gamma'(1) = -\gamma.$$

44. При $k \ge 0$ имеем $r_k(m) \sim \frac{1}{2}(2m)^{(k+1)/2}\Gamma((k+1)/2) - \delta_{k0} - \sum_{j\ge0}(-1)^j B_{k+2j+1}/\,((k+2j+1)j!\,(2m)^j)$. При $k=-1$ вклады от $f_k^{(j-1)}(m)$ в (36) взаимно уничтожаются с подобными членами в разложении H_{m-1}, и имеем $r_{-1}(m) = H_{m-1} + (1/\sqrt{2m})\sum_{t\ge0} f_{-1}(t) \sim \frac{1}{2}(\ln(2m)+\gamma) - \sum_{j\ge1}(-1)^j B_{2j}/(2j)j!\,(2m)^j$. Поэтому вклад в W_{m-1} члена N^t/t (33) получается из суммы $m\sum_{t\ge1} t^{-1}\exp(-t^2/2m)(1-t^3/3m^2+t^6/18m^4)(1-t^4/4m^3)(1-t/2m-t^2/8m^2) + O(m^{-1/2}) = \frac{1}{2}m\ln m + \frac{1}{2}(\ln 2+\gamma)m - \frac{5}{12}\sqrt{2\pi m} + \frac{4}{9} + O(m^{-1/2})$. Член $-\frac{1}{2}N^{t-1}$ дает вклад

* Имеется русский перевод: Дейкстра Э. В. *Дисциплина программирования*. — М.: Мир, 1978. — *Прим. перев.*

$-\frac{1}{2}\sum_{t\ge1}\exp(-t^2/2m)(1-t^3/3m^2)(1-t/2m)(1+t/m)+O(m^{-1/2})=-\frac{1}{4}\sqrt{2\pi m}+\frac{1}{3}$. Член $\frac{1}{2}\delta_{t1}$ дает вклад $\frac{1}{2}$. И наконец, член $\frac{1}{2}(t-1)B_2N^{t-2}$ дает вклад $\frac{1}{12}m^{-1}\sum_{t\ge1}t\exp(-t^2/2m)+O(m^{-1/2})=\frac{1}{12}+O(m^{-1/2})$.

45. Рассуждение, использованное при выводе тождества (42), справедливо и для тождества (43), нужно только отбросить вычеты в точках $z=-1$ и $z=0$.

46. Действуя так же, как при вычислении интеграла (45), получаем $(s-1)!/\ln 2+\delta_s(n)$, где

$$\delta_s(n)=\frac{2}{\ln 2}\sum_{k\ge1}\Re(\Gamma(s-2\pi ik/\ln 2)\exp(2\pi ik\lg n)).$$

[Обратите внимание на то, что $|\Gamma(s+it)|^2=(\prod_{0\le k<s}(k^2+t^2))\pi/(t\sinh\pi t)$, где $s\ge0$ — целое, так что можно найти верхнюю границу для функции $\delta_s(n)$.]

47. Сумма $\sum_{j\ge1}e^{-n/2^j}(n/2^j)^s$ на самом деле равна интегралу из упр. 46 при всех $s>0$.

48. Воспользовавшись промежуточным тождеством

$$1-e^{-x}=\frac{-1}{2\pi i}\int_{-1/2-i\infty}^{-1/2+i\infty}\Gamma(z)x^{-z}\,dz,$$

действуем так же, как описано в тексте, но теперь $1-e^{-x}$ выполняет роль $e^{-x}-1+x$; $V_{n+1}/(n+1)=(-1/2\pi i)\int_{-1/2-i\infty}^{-1/2+i\infty}\Gamma(z)n^{-z}\,dz/(2^{-z}-1)+O(n^{-1})$, а интеграл равен $\lg n+\gamma/\ln 2-\frac{1}{2}-\delta_0(n)$ в обозначениях упр. 46. (Таким образом, величина A_N из упр. 38 равна $N(1/\ln 2-\delta_0(N-1)-\delta_{-1}(N))+O(1)$.)

49. Правую часть равенства (40) можно улучшить, заменив ее на $e^{-x}(n/x+\frac{1}{2}x+x^3O(n^{-1}))$. Результат выразится в том, что будет вычтена сумма из упр. 47 с коэффициентом $\frac{1}{2}$ и $O(1)$ в (47) заменится выражением $2-\frac{1}{2}(1/\ln 2+\delta_1(n))+O(n^{-1})$. (Двойка появилась из "$2/n$" в (45).)

50. $U_{mn}=n\log_m n+n((\gamma-1)/\ln m-\frac{1}{2}+\delta_{-1}(n))+m/(m-1)-1/(2\ln m)-\frac{1}{2}\delta_1(n)+O(n^{-1})$, где $\delta_s(n)$ определяется в упр. 46 но $\ln 2$ и $\lg$ заменяются на $\ln m$ и $\log_m$. [*Замечание.* При $m=2$, 3, 4, 5, 10, 100, 1000 и 10^6 имеем $\delta_{-1}(n)<$ .0000001725, .00041227, .000296, .00085, .00627, .068, .153, .341 соответственно.]

51. Пусть $N=2m$. Можно распространить суммирование в (35) на все значения $t\ge1$. Тогда сумма будет равна

$$\sum_{t\ge1}\frac{1}{2\pi i}\int_{a-i\infty}^{a+i\infty}\Gamma(z)(t^2/N)^{-z}t^k\,dz=\frac{1}{2\pi i}\int_{a-i\infty}^{a+i\infty}\Gamma(z)N^z\zeta(2z-k)\,dz$$

при условии, что $a>(k+1)/2$. Итак, необходимо знать свойства дзета-функции. Если $\Re(w)\ge-q$, то $\zeta(w)=O(|w|^{q+1})$ при $|w|\to\infty$; следовательно, контур интегрирования можно как угодно далеко смещать влево, если только учесть вычеты. Сомножитель $\Gamma(z)$ имеет полюсы в точках $0,-1,-2,\ldots$, а $\zeta(2z-k)$ имеет полюс лишь в точке $z=(k+1)/2$. Вычет в точке $z=-j$ равен $N^{-j}(-1)^j\zeta(-2j-k)/j!$, а $\zeta(-n)=(-1)^nB_{n+1}/(n+1)$. Вычет в точке $z=(k+1)/2$ равен $\frac{1}{2}\Gamma((k+1)/2)N^{(k+1)/2}$. Но если $k=-1$, то в точке $z=0$ имеется двойной полюс, а $\zeta(z)=1/(z-1)+\gamma+O(|z-1|)$, так что вычет в 0 в этом случае равен $\gamma+\frac{1}{2}\ln N-\frac{1}{2}\gamma$. Таким образом получаем асимптотический ряд, упомянутый в ответе к упр. 44.

52. Положим, что $x=t/n$; тогда

$$\binom{2n}{n+t}\Big/\binom{2n}{n}=\exp(-2n(x^2/1\cdot2+x^4/3\cdot4+\cdots)+(x^2/2+x^4/4+\cdots)$$
$$-(1/6n)(x^2-x^4+\cdots)+\cdots).$$

Искомую сумму теперь можно представить в виде ряда $\sum_{t\ge1} t^k d(t) e^{-t^2/n}$ при различных k. Поскольку $\zeta(z)^2 = \sum_{t\ge1} d(t) t^{-z}$, то, действуя так же, как и в упр. 51, можно вычислить вычеты функции $\Gamma(z) n^z \zeta(2z-k)^2$ при $k \ge 0$. Вычет в точке $z = -j$ равен

$$n^{-j}(-1)^j (B_{2j+k+1}/(2j+k+1))^2/j!\,,$$

а вычет в точке $z = (k+1)/2$ равен $n^{(k+1)/2}\Gamma((k+1)/2)(\gamma + \frac14 \ln n + \frac14 \psi((k+1)/2))$, где $\psi(z) = \Gamma'(z)/\Gamma(z) = H_{z-1} - \gamma$. Так, например, если $k = 0$, то $\sum_{t\ge1} e^{-t^2/n} d(t) = \frac14\sqrt{\pi n}\ln n + (\frac34\gamma - \frac12 \ln 2)\sqrt{\pi n} + \frac14 + O(n^{-M})$ при любом M. Чтобы получить $S_n/\binom{2n}{n}$, добавьте к этой величине $(\frac{1}{32}\ln n + \frac{3}{32}\gamma + \frac{1}{24} - \frac{1}{16}\ln 2)\sqrt{\pi/n} + O(n^{-1})$. (См. упр. 1.2.7–23 и 1.2.9–19.)

53. Пусть $q = 1 - p$. Обобщая анализ, выполненный в упр. 36, (c), получим, что если

$$x_n = a_n + \sum_{k\ge2} \binom{n}{k} (p^k q^{n-k} + q^k p^{n-k}) x_n,$$

то

$$x_n = a_n + \sum_{k\ge2} \binom{n}{k} (-1)^k \hat a_k (p^k + q^k)/(1 - p^k - q^k).$$

Следовательно, можно, как и прежде, найти величины B_N и C_N. Множитель $\frac14$ в B_N нужно заменить на pq. Анализ асимптотического выражения для U_N выполняется, по существу, так же, как в тексте, причем

$$\begin{aligned} T_n &= \sum_{r\ge1,\, s\ge0} \binom{r}{s} (e^{-np^s q^{r-s}} - 1 + np^s q^{r-s}) \\ &= \frac{1}{2\pi i} \int_{-3/2-i\infty}^{-3/2+i\infty} \Gamma(z) n^{-z} (p^{-z} + q^{-z})\, dz/(1 - p^{-z} - q^{-z}) \\ &= (n/h_p)(\ln n + \gamma - 1 + h_p^{(2)}/2h_p - h_p + \delta(n)) + O(1), \end{aligned}$$

где $h_p = -(p\ln p + q\ln q)$, $h_p^{(2)} = p(\ln p)^2 + q(\ln q)^2$, а $\delta(n) = \sum \Gamma(z) n^{-1-z}/h_p$, причем суммирование выполняется по всем комплексным числам $z \ne 1$, таким, что $p^{-z} + q^{-z} = 1$. Это последнее множество точек, по-видимому, трудно исследовать в общем случае, но при $p = \phi^{-1}$, $q = \phi^{-2}$ решением будут точки $z = (-1)^{k+1} + k\pi i/\ln\phi$. Главный член $(n\ln n)/h_p$ также можно было бы получить из общей формулы ван Эмдена, приведенной в ответе к упр. 29. При $p = \phi^{-1}$ имеем $1/h_p \approx 1.503718$, по сравнению с $1/h_{1/2} \approx 1.442695$.

54. Пусть C — окружность радиуса $(M + \frac12)b$ и интеграл по C стремится к нулю при $M \to \infty$. (Асимптотическое выражение для U_n можно теперь вывести по-новому, разлагая $\Gamma(n+1)/\Gamma(n+ibm)$. Метод, рассматриваемый в этом упражнении, применим ко *всем* суммам вида

$$\sum_k \binom{n}{k} (-1)^{n-k} f(k) = \frac{-1}{2\pi i} \oint B(n+1, -z) f(z)\, dz,$$

если функция f выбрана обоснованно!) Последняя формула выведена в работе N. E. Nörlund, *Vorlesungen Über Differenzenrechnung* (Berlin: Springer, 1924), §103.)

55. Замените строки 04–06 в программе Q следующим образом.

```
2H ENTA 0,2                  STA  INPUT,3  c≤b<a        JGE  5F
   INCA 0,3                  STX  INPUT,2               CMPX INPUT,4   a<b,c
   SRB  1                 5H LDA  INPUT,4  rA←b         JGE  5B
   STA  *+1(0:2)             JMP  6F                    LDA  INPUT,3   a<c<b
   ENT4 *                 4H LDA  INPUT,3  b<c≤a        LDX  INPUT,4
   LDA  INPUT,2  rA←a        LDX  INPUT,2               STX  INPUT,3
   LDX  INPUT,3  rX←c        STX  INPUT,3               JMP  6F
   CMPA INPUT,3              JMP  5F                 5H LDX  INPUT,4   b≤a<c
   JL   1F                3H STX  INPUT,2  c≤a≤b        STX  INPUT,2
   CMPA INPUT,4  rA:b        LDX  INPUT,4            6H LDX  INPUT+1,2
   JLE  3F                   STX  INPUT,3               STX  INPUT,4
   CMPX INPUT,4  rX:b        JMP  6F                    ENT4 2,2
   JG   4F                1H CMPA INPUT,4               ENT5 0,3
```

После этого должно следовать `STA INPUT+1,2` (см. замечание после (27)). Замените команду в строке 22 командой `STX INPUT+1,2`. Если в системе команд отсутствует команда поразрядного сдвига, то первые три команды в программе нужно заменить командой `ENTX 0,2`. `INCX 0,3`; `ENTA 0`; `DIV =2=`.

Сущность этой программы состоит в организации обмена R_{i+1} с $R_{\lfloor (l+r)/2 \rfloor}$ и сортировки записей R_l, R_{l+1} и R_r. Затем к $R_{l+1} \ldots R_{r-1}$ применяется обычная процедура разбиения. Несколько команд можно сэкономить, просто поместив медианный элемент в регистр rA и переписав R_l туда, где ранее был медианный элемент. Далее все нужно продолжить, как в программе Q. Но такой подход может привести к нежелательным последствиям, поскольку требуется порядка N^2 шагов для сортировки массива $N\ N{-}1\ \ldots\ 1$. (Этот неожиданный побочный результат впервые подметил Д. Б. Колдрик (D. B. Coldrick), но его нужно проверить самостоятельно. Попробуйте!) Рекомендуемый выше метод, авторство которого принадлежит Р. Седгевику, оказался свободным от связанной с длиной слова аномалией, но работает так же быстро. При использовании такой схемы не нужно проверять K_0 и K_{N+1}. Следовательно, проверка граничных значений не сдерживает скорости выполнения внутреннего цикла.

56. Можно решить рекуррентное соотношение $\binom{n}{3}x_n = b_n + 2\sum_{k=1}^{n}(k-1)(n-k)x_{k-1}$ при $n > m$, положив $y_n = nx_n$, $u_n = ny_{n+1} - (n+2)y_n$, $v_n = nu_{n+1} - (n-5)u_n$. Отсюда $v_n = 6(b_{n+2} - 2b_{n+1} + b_n)$ при $n > m$. *Например, пусть* $x_n = \delta_{n1}$ *при* $n \le m$ *и пусть* $b_n \equiv 0$. Тогда $v_n = 0$ при всех $n > m$; следовательно, $n^{\underline{5}}u_{n+1} = m^{\underline{5}}u_{m+1}$. Поскольку $y_{m+1} = 12/m$ и $y_{m+2} = 12/(m+1)$, окончательно находим $x_n = \frac{48}{7}(n+1)/m(m+1)(m+2) + \frac{36}{7}(m-1)^{\underline{4}}/n^{\underline{6}}$ при $n > m$. В общем случае пусть $f_n = \big(12/(n-1)(n-2)\big)\sum_{k=1}^{n}(k-1)(n-k)x_{k-1}$; если b_n тождественно равно 0, то решением при $n > m$ будет

$$x_n = (n+1)\frac{(m+1)f_{m+2} - (m-4)f_{m+1}}{7(m+1)(m+2)} - \frac{\big((m+1)f_{m+2} - (m+3)f_{m+1}\big)m^{\underline{5}}}{7n^{\underline{6}}}.$$

Если $b_n = \binom{n}{3}/n^{\underline{p}}$ и $x_n = 0$ при $n \le m$, то решением будет

$$\frac{x_n}{n+1} = \frac{(p-3)(p-2)}{(p-6)(p+1)(n+1)^{\underline{p+1}}} + \frac{12}{7}\frac{1}{(p+1)(m+2)^{\underline{p+1}}} - \frac{12}{7}\frac{(m+1-p)^{\underline{6-p}}}{(p-6)(n+1)^{\underline{7}}}$$

при $n > m$. За исключением случаев, когда $p = -1$, получим $x_n/(n+1) = \frac{12}{7}(H_{n+1} - H_{m+2}) + \frac{37}{49} + \frac{12}{49}(m+2)^{\underline{7}}/(n+1)^{\underline{7}}$, а когда $p = 6$, $x_n/(n+1) = -\frac{12}{7}(H_{n-6} - H_{m-5})/(n+1)^{\underline{7}} + \frac{12}{49}/(m+2)^{\underline{7}} + \frac{37}{49}/(n+1)^{\underline{7}}$.

Рассуждая, как в упр. 21–23, придем к выводу, что первая фаза разделения теперь внесет 1 в A, t в B и $N-1$ в C, где t определяется, как и ранее, но *после* перекомпоновки, которая сделана в упр. 55. При новых предположениях $b_{stN} = 6\binom{s-2}{t}\binom{N-s-1}{t}/N\binom{N-1}{s-1}$.

Следовательно, рекуррентные соотношения, выведенные выше, в этой задаче появляются следующим образом.

	Значения для $N \le M$	$b_N/\binom{N}{3}$ для $N > M$	Решения для $N > M$
A_N	0	1	$(N+1)(\frac{12}{7}/(M+2))-1+O(N^{-6})$
B_N	0	$(N-4)/5$	$(C_N-3A_N)/5$
C_N	0	$N-1$	$(N+1)(\frac{12}{7}(H_{N+1}-H_{M+2})+\frac{37}{49}-\frac{24}{7}/(M+2))+2+O(N^{-6})$
D_N	$N-H_N$	0	$(N+1)(1-\frac{12}{7}H_{M+1}/(M+2)-\frac{4}{7}/(M+2))+O(N^{-6})$
E_N	$N(N-1)/4$	0	$(N+1)(\frac{6}{35}M-\frac{17}{35}+\frac{6}{7}/(M+2))+O(N^{-6})$

Аналогично $S_N = \frac{3}{7}(N+1)(5M+3)/(2M+3)(2M+1)-1+O(N^{-6})$. В среднем суммарное время выполнения программы в упр. 55 равно $53\frac{1}{2}A_N + 11B_N + 4C_N + 3D_N + 8E_N + 9S_N + 7N$. Выбор $M = 9$ лишь немного лучше, чем $M = 10$, и приводит к среднему времени выполнения, равному примерно $10\frac{22}{35}N\ln N + 2.116N$ [*Acta Inf.* **7** (1977), 336–341]. После замены SRB на DIV $11A_N$ добавляется к среднему времени выполнения и требует установить $M = 10$.

РАЗДЕЛ 5.2.3

1. Нет, но метод, в котором используется ∞ (описанный непосредственно перед алгоритмом S), устойчив.

2. Просмотр линейного списка, хранящегося в последовательных ячейках памяти, часто выполняется несколько быстрее, если двигаться в сторону убывания индексов, поскольку, как правило, легче проверить, равен ли индекс 0, чем проверить, превышает ли он N. (По той же причине при поиске на шаге S2 индекс убывает от j к 1; тем не менее см. упр. 8!)

3. (a) Перестановка $a_1 \ldots a_{N-1}N$ соответствует исходным массивам

$$Na_2\ldots a_{N-1}a_1,\quad a_1Na_3\ldots a_{N-1}a_2,\quad \ldots,\quad a_1a_2\ldots a_{N-2}Na_{N-1},\quad a_1\ldots a_{N-1}N.$$

(b) Как показано в разделе 1.2.10, на первой итерации шага S2 число случаев изменения максимума равно $H_N - 1$. [Следовательно, B_N можно найти из соотношения 1.2.7–(8).]

4. Если исходный файл является перестановкой множества $\{1, 2, \ldots, N\}$, то число случаев, когда на шаге S3 окажется $i = j$, в точности на единицу меньше числа циклов в этой перестановке. (В самом деле, нетрудно показать, что на шагах S2 и S3 элемент j попросту удаляется из своего цикла. Следовательно, шаг S3 бесполезен лишь в том случае, если элемент j был наименьшим в своем цикле.) Согласно равенствам (1.3.3–(21)) можно было бы сэкономить в среднем $H_N - 1$ из $N - 1$ выполнений шага S3.

Таким образом, было бы неэффективно вставлять перед шагом S3 дополнительную проверку "$i = j$?". Вместо того чтобы сравнивать i с j, можно слегка удлинить программу для шага S2, продублировав часть команд, чтобы не переходить к шагу S3, если первоначальный выбор K_j не изменился во время поиска максимума. За счет этого программа S стала бы работать чуть-чуть быстрее.

5. $(N-1) + (N-3) + \cdots = \lfloor N^2/4 \rfloor$.

6. (a) Если на шаге S3 окажется, что $i \ne j$, то после его выполнения число инверсий уменьшится на $2m - 1$, где m на единицу больше числа ключей, лежащих в интервале между K_i и K_j среди $K_{i+1} \ldots K_{j-1}$. Ясно, что m не меньше, чем вклад в значение параметра B на предыдущем шаге S2. Примените теперь следствие из упр. 4, связывающее циклы с условием $i = j$. (b) Любую перестановку можно получить из $N \ldots 2\ 1$ путем последовательных взаимообменов соседних элементов, которые нарушают исходное упорядочение. (Примените в обратной последовательности те обмены, под действием которых

перестановка сортируется в порядке убывания.) Каждая такая операция уменьшает I на единицу и изменяет C на ± 1. Следовательно, ни для одной перестановки значение $I - C$ не превосходит соответствующего значения для $N \ldots 2\ 1$. [Из упр. 5 вытекает, что неравенство $B \le \lfloor N^2/4 \rfloor$ — наилучшее из возможных.]

7. В работе A. C. Yao, *On straight selection sort* (Computer Science Technical Report 185, Princeton University, 1988), показано, что дисперсия равна $\alpha N^{1.5} + O(N^{1.495} \log N)$, где $\alpha = \frac{4}{3}\sqrt{\pi} \ln \frac{4}{e} \approx 0.9129$. В ней также высказано предположение, что в действительности составляющая ошибки значительно меньше.

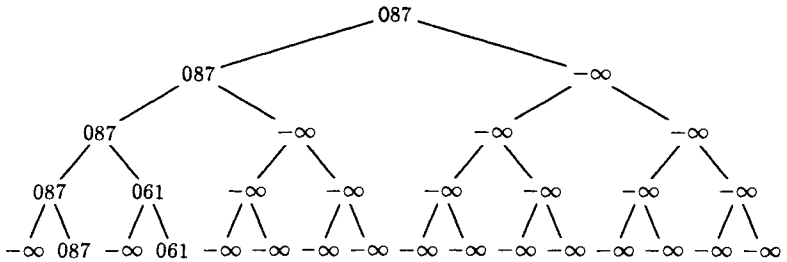
8. Можно начинать очередную итерацию шага S2 с позиции K_i, поскольку мы запомнили $\max(K_1, \ldots, K_{i-1})$. Один из способов учета всей этой дополнительной информации — использование таблицы связи $L_1 \ldots L_N$, такой, что K_{L_k} равно предыдущему выделенному элементу всякий раз, когда K_k также выделено; $L_1 = 0$. [Тот же результат можно получить, используя меньший объем дополнительной памяти, "заплатив" за это лишней операцией сравнения.]

В приведенной ниже MIX-программе применяется модификация адреса в ходе выполнения, которая ускоряет реализацию внутреннего цикла. Исходное состояние регистров таково: rI1 $\equiv j$, rI2 $\equiv k - j$, rI3 $\equiv i$, rA $\equiv K_i$.

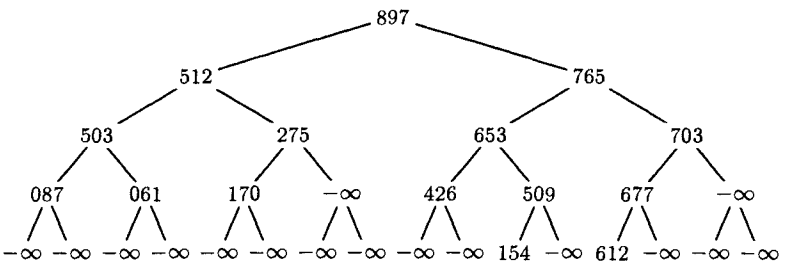
01	START	ENT1	N	1	$j \leftarrow N$.	
02		STZ	LINK+1	1		
03		JMP	9F	1		
04	1H	ST1	6F(0:2)	$N - D$	Модификация адреса во внутреннем цикле.	
05		ENT4	INPUT,1	$N - D$		
06		ST4	7F(0:2)	$N - D$		
07		ENT4	LINK,1	$N - D$		
08		ST4	8F(0:2)	$N - D$		
09	7H	CMPA	INPUT+J,2	A		[Адрес модифицирован]
10		JGE	*+4	A	Переход, если $K_i \ge K_k$.	
11	8H	ST3	LINK+J,2	$N + 1 - C$	Иначе — $L_k \leftarrow i$.	[Адрес модифицирован]
12	6H	ENT3	J,2	$N + 1 - C$	$i \leftarrow k$.	[Адрес модифицирован]
13		LDA	INPUT,3	$N + 1 - C$		
14		INC2	1	A	$k \leftarrow k + 1$.	
15		J2NP	7B	A	Переход, если $k \le j$.	
16	4H	LDX	INPUT,1	N		
17		STX	INPUT,3	N	$R_i \leftarrow R_j$.	
18		STA	INPUT,1	N	$R_j \leftarrow$ предыдущее значение R_i.	
19		DEC1	1	N	$j \leftarrow j - 1$.	
20		ENT2	0,3	N	rI2 $\leftarrow i$.	
21		LD3	LINK,3	N	$i \leftarrow L_i$.	
22		J3NZ	5F	N	Если $i > 0$, k начнет с i.	
23	9H	ENT3	1	C	Иначе — $i \leftarrow 1$.	
24		ENT2	2	C	k начнет с 2.	
25	5H	DEC2	0,1	$N + 1$		
26		LDA	INPUT,3	$N + 1$	rA $\leftarrow K_i$.	
27		J2NP	1B	$N + 1$	Переход, если $k \le j$.	
28		J1P	4B	$D + 1$	Переход, если $j > 0$. ▌	

9. $N - 1 + \sum_{N \ge k \ge 2}((k-1)/2 - 1/k) = \frac{1}{2}\binom{N}{2} + N - H_N$. [Средние значения параметров C и D равны $H_N + 1$ и $H_N - \frac{1}{2}$ соответственно. Следовательно, среднее время выполнения программы равно $(1.25N^2 + 31.75N - 15H_N + 14.5)u$.] Программа H работает значительно лучше.

10.



11.



12. $2^n - 1$, один раз для каждого $-\infty$ в узле ветвления.

13. Если $K \geq K_{r+1}$, то после шага H4 можно переходить к шагу H5 при $j = r$. (На шаге H5 ничего не делается, если не выполнено условие $K_r < K_{r+1}$; в этом случае на шаге H6 обязательно произойдет переход к шагу H8.) Чтобы на протяжении всего алгоритма обеспечить выполнение условия $K \geq K_{r+1}$, можно начать с $K_{N+1} \leq \min(K_1, \ldots, K_N)$. Вместо присвоения $R_r \leftarrow R_1$ на шаге H2 установим $R_{r+1} \leftarrow R_{N+1}$ и $R_{N+1} \leftarrow R_1$; установим также $R_2 \leftarrow R_{N+1}$ после того, как выполнится условие $r = 1$. (Этот прием не ускоряет алгоритм и не укорачивает программу H.)

14. Чтобы получить простую очередь (или стек), вставляя элемент, приписывайте ему ключ, меньший (соответственно больший) всех ранее присвоенных ключей.

15. Так как преследовалась цель повышения эффективности, следующее решение несколько замысловато: в нем пропускаются все числа, кратные 3 [*CACM* **10** (1967), 570].

P1. Установить $p[1] \leftarrow 2$, $p[2] \leftarrow 3$, $k \leftarrow 2$, $n \leftarrow 5$, $d \leftarrow 2$, $r \leftarrow 1$, $t \leftarrow 25$ и поместить в приоритетную очередь элемент $(25, 10, 30)$. (В этом алгоритме $p[i]$ — i-е простое число, k — количество найденных до сих пор простых чисел, n — кандидат в простые числа, d — расстояние до следующего кандидата, r — число элементов в очереди, $t = p[r+2]^2$ — следующее значение n, для которого надо увеличить r. Элементы очереди имеют вид $(u, v, 6p)$, где p — наименьший простой делитель u, $v = 2p$ или $4p$ и $u + v$ не делится на 3.)

P2. Пусть (q, q', q'') — элемент очереди с наименьшей первой компонентой. Замените его в очереди на $(q + q', q'' - q', q'')$. (Это означает следующий множитель $q''/6$, который должен быть исключен.) Если $n > q$, повторять этот шаг до тех пор, пока не станет $n \leq q$.

P3. Если $n > N$, прекратить выполнение алгоритма. В противном случае, если $n < q$, установить $k \leftarrow k + 1$, $p[k] \leftarrow n$, $n \leftarrow n + d$, $d \leftarrow 6 - d$ и повторить этот шаг.

P4. (Теперь $n = q$ — непростое число.) Если $n = t$, установить $r \leftarrow r+1$, $u \leftarrow p[r+2]$, $t \leftarrow u^2$ и вставить в очередь либо $(t, 2u, 6u)$, либо $(t, 4u, 6u)$, если $u \bmod 3 = 2$ или $u \bmod 3 = 1$ соответственно.

P5. Установить $n \leftarrow n + d$, $d \leftarrow 6 - d$ и вернуться к шагу P2. ▌

Начало вычислений выглядит следующим образом.

Содержимое очереди	Найденные простые числа
(25, 10, 30)	5, 7, 11, 13, 17, 19, 23
(35, 20, 30)(49, 28, 42)	29, 31
(49, 28, 42)(55, 10, 30)	37, 41, 43, 47
(55, 10, 30)(77, 14, 42)(121, 22, 66)	53

Если очередь реализована в виде пирамиды, то можно найти все простые числа $\leq N$ за $O(N \log N)$ шагов; размер пирамиды не превышает количества простых чисел $\leq \sqrt{N}$. Решето Эратосфена (реализация приведена в упр. 4.5.4–8) — метод с временем выполнения порядка $O(N \log\log N)$, требующий значительно больше памяти с произвольным доступом. Более эффективные методы реализации будут рассмотрены в разделе 7.1.

16. **I1.** Установить $K \leftarrow$ вставляемый ключ; $j \leftarrow n + 1$.

I2. Установить $i \leftarrow \lfloor j/2 \rfloor$.

I3. Если $i = 0$ или $K_i \geq K$, то установить $K_j \leftarrow K$ и прекратить выполнение алгоритма.

I4. Установить $K_j \leftarrow K_i$, $j \leftarrow i$ и возвратиться к шагу I2. ▌

[В работе T. Porter, I. Simon, *IEEE Trans.* **SE-1** (1975), 292–298, показано, что если задана случайная пирамида из равномерно распределенных чисел и A_{n+1} — среднее число случаев выполнения шага 4, то получится $A_n = \lfloor \lg n \rfloor + (1 - n^{-1})A_{n'}$ при $n > 1$, где $n = (1b_{l-1}b_{l-2}\ldots b_0)_2$ влечет за собой $n' = (1b_{l-2}\ldots b_0)_2$. Если $l = \lfloor \lg n \rfloor$, эта величина всегда $\geq A_{2^{l+1}-1} = (2^{l+1} - 2)/(2^{l+1} - 1)$ и всегда $\leq A_{2^l} < \alpha$, где α — константа в (19).]

17. Алгоритм H преобразует массив 1 2 3 в пирамиду 3 2 1, а алгоритм из упр. 16 преобразует его в пирамиду 3 1 2. [*Замечание.* Время выполнения этого последнего алгоритма построения пирамиды в наихудшем случае составляет порядка $N \log N$, но эксперименты показали, что для случайного исходного массива шаг 2 во время построения пирамиды выполняется всего около $2.28N$ раз. В работе R. Hayward, C. McDiarmid, *J. Algorithms* **12** (1991), 126–153, строго доказано, что значения постоянных коэффициентов пропорциональности лежат между 2.2778 и 2.2994.]

18. Удалите шаг H6 и замените операции на шаге H8 следующими операциями.

H8′. [Продвинуться обратно вверх.] Установить $j \leftarrow i$, $i \leftarrow \lfloor j/2 \rfloor$.

H9′. [Подходит ли K?] Если $K \leq K_i$ или $j = l$, установить $R_j \leftarrow R$ и возвратиться к шагу H2. В противном случае установить $R_j \leftarrow R_i$ и возвратиться к шагу H8′. ▌

Это, по существу, тот же метод, что и в упр. 16, но с другой начальной точкой в пирамиде. Массив изменяется так же, как в алгоритме H. Эмпирические проверки этого метода показывают, что число присвоений $R_j \leftarrow R_i$, приходящихся на одну операцию протаскивания в фазе выбора, равно $(0, 1, 2)$ с вероятностями (.837, .135, .016) соответственно. Этот метод несколько удлиняет программу H, но повышает асимптотическую скорость до $13N \lg N + O(N)$. Желательно иметь в системе команд MIX команду деления пополам содержимого индексного регистра.

В работе C. J. H. McDiarmid, B. A. Reed, *J. Algorithms* **10** (1989), 352–365, доказано, что при такой модификации сохраняется в среднем $(3\beta - 8)N \approx 0.232N$ сравнений на этапе формирования пирамиды, где β определено в ответе к упр. 27. Более углубленный

анализ модификаций Флойда приведен в работе I. Wegener, *Theoretical Comp. Sci.* **118** (1993), 81–98.

В работе J. Wu, H. Zhu, *J. Comp. Sci. и Tech.* **9** (1994), 261–266, высказано соображение, что может быть использован и метод двоичного поиска , так что каждая операция протаскивания на фазе выбора будет включать не более $\lg N + \lg\lg N$ сравнений и $\lg N$ перемещений записей.

19. Действуйте так же, как в измененном алгоритме протаскивания (упр. 18), установив значения $K = K_N$, $l = 1$ и $r = N - 1$ с заданного значения j на шаге H3.

20. При $0 \le k \le n$ количество положительных целых чисел $\le N$, двоичное представление которых имеет вид $(b_n \ldots b_k a_1 \ldots a_q)_2$ при некотором $q \ge 0$, равно, очевидно, $(b_{k-1} \ldots b_0)_2 + 1 + \sum_{0 \le q < k} 2^q = (1b_{k-1} \ldots b_0)_2$.

21. Пусть $j = (c_r \ldots c_0)_2$ принадлежит интервалу $\lfloor N/2^{k+1} \rfloor = (b_n \ldots b_{k+1})_2 < j < (b_n \ldots b_k)_2 = \lfloor N/2^k \rfloor$. Тогда s_j равно количеству положительных целых чисел $\le N$, двоичное представление которых — суть $(c_r \ldots c_0 a_1 \ldots a_q)_2$ при определенном $q \ge 0$, а именно — $\sum_{0 \le q < k} 2^q = 2^{k+1} - 1$. Следовательно, число неособых деревьев размером $2^{k+1} - 1$ равно

$$\lfloor N/2^k \rfloor - \lfloor N/2^{k+1} \rfloor - 1 = \lfloor (N - 2^k)/2^{k+1} \rfloor.$$

[Для вывода последнего тождества воспользуйтесь репликативным законом из упр. 1.2.4–38 при $n = 2$ и $x = N/2^{k+1}$.]

22. До того как станет $l = 1$, пять возможных вариантов таковы: 5 3 4 1 2, 3 5 4 1 2, 4 3 5 1 2, 1 5 4 3 2 и 2 5 4 1 3. Каждый из этих вариантов $a_1 a_2 a_3 a_4 a_5$ приводит к трем перестановкам $(a_1 a_2 a_3 a_4 a_5,\ a_1 a_4 a_3 a_2 a_5,\ a_1 a_5 a_3 a_4 a_2)$, возможным до того, как получим $l = 2$.

23. (a) После B итераций имеем $j \ge 2^B l$; следовательно, $2^B l \le r$. (b) Имеем

$$\sum_{l=1}^{n} \lfloor \log_2 (N/l) \rfloor = (\lfloor N/2 \rfloor - \lfloor N/4 \rfloor) + 2(\lfloor N/4 \rfloor - \lfloor N/8 \rfloor) + 3(\lfloor N/8 \rfloor - \lfloor N/16 \rfloor) + \cdots =$$
$$= \lfloor N/2 \rfloor + \lfloor N/4 \rfloor + \lfloor N/8 \rfloor + \cdots = N - \nu(N),$$

где $\nu(N)$ — число единиц в двоичном представлении числа N. Из упр. 1.2.4–42 получаем также $\sum_{r=1}^{N-1} \lfloor \lg r \rfloor = N \lfloor \lg N \rfloor - 2^{\lfloor \lg N \rfloor + 1} + 2$. Из теоремы H известно, что эта верхняя оценка для величины B — наилучшая из возможных на фазе построения пирамиды. Интересно, кроме того, отметить, что существует единственная пирамида из ключей $\{1, 2, \ldots, N\}$, такая, что в течение всей фазы выбора в алгоритме H значение K тождественно равняется 1. (При $N = 7$, например, такой пирамидой будет 7 5 6 2 4 3 1; нетрудно осуществить переход от N к $N + 1$.) На этой пирамиде и достигается максимальное значение B на фазе выбора пирамидальной сортировки (так же, как и максимальное значение $\lfloor N/2 \rfloor$ параметра D). Значит, верхняя граница значения параметра B, т. е. наилучшее значение из возможных для всей сортировки, равно $N - \nu(N) + N \lfloor \lg N \rfloor - 2^{\lfloor \lg N \rfloor + 1} + 2$.

24. $\sum_{k=1}^{N} \lfloor \lg k \rfloor^2 = (N + 1 - 2^n) n^2 + \sum_{0 \le k < n} k^2 2^k = (N + 1) n^2 - (2n - 3) 2^{n+1} - 6$, где $n = \lfloor \lg N \rfloor$ (см. упр. 4.5.2–22). Следовательно, дисперсия для последнего протаскивания есть $\beta_N = ((N+1)n^2 - (2n-3)2^{n+1} - 6)/N - ((N+1)n + 2 - 2^{n+1})^2/N^2 = O(1)$. Стандартное отклонение величины B'_N равно $\left(\sum\{\beta_s \mid s \in M_N\}\right)^{1/2} = O(\sqrt{N})$.

25. Протаскивание "равномерно", и каждое сравнение $K_j : K_{j+1}$ с вероятностью $\frac{1}{2}$ дает результат $<$. Средний вклад в значение параметра C в этом случае равен в точности половине суммы средних вкладов в значения параметров A и B, а именно — $((2n-3)2^{n-1} + \frac{1}{2})/(2^{n+1} - 1)$.

26. (a) $(\frac{10}{25}+\frac{1}{2}+1\frac{3}{9}+\frac{1}{2}+1\frac{1}{2}+1\frac{2}{5}+2\frac{1}{2}+\frac{1}{2}+1\frac{1}{2}+1\frac{1}{2}+2\frac{1}{2}+1\frac{1}{2}+2+2+3+0+1+1+2+1+2+2+3+1+2+2)/26 = 1189/780 \approx 1.524$.

(b) $(\sum_{k=1}^{N}\nu(k) - N + \frac{1}{2}\lfloor N/2\rfloor - \frac{1}{2}n + \sum_{k=1}^{n-1}\min(\alpha_{k-1},\, \alpha_k - \alpha_{k-1} - 1)/(\alpha_k - 1))/N$, где $\nu(k)$ — количество единиц в двоичном представлении числа k, а $\alpha_k = (1b_k\ldots b_0)_2$. Если $N = 2^{e_1} + 2^{e_2} + \cdots + 2^{e_t}$, где $e_1 > e_2 > \cdots > e_t \geq 0$, можно показать, что $\sum_{k=0}^{N}\nu(k) = \frac{1}{2}((e_1+2)2^{e_1} + (e_2+4)2^{e_2} + \cdots + (e_t+2t)2^{e_t}) + t - N$. [Асимптотические свойства этой суммы можно проанализировать при помощи преобразования Меллина; см. Flajolet, Grabner, Kirschenhofer, Prodinger, Tichy, *Theoretical Comp. Sci.* **123** (1994), 291–314.]

27. Дж. У. Ренч (мл.) (J. W. Wrench, Jr.) пришел к выводу, что в общем случае ряд Ламберта $\sum_{n\geq 1} a_n x^n/(1-x^n)$ можно представить в виде $\sum_{N\geq 1}(\sum_{d\backslash N} a_d)x^N = \sum_{m\geq 1}(a_m + \sum_{k\geq 1}(a_m + a_{m+k})x^{km})\,x^{m^2}$.

[На случай, когда $a_n = 1$ и $a_n = n$, первым обратил внимание И. Г. Ламберт (J. H. Lambert) в работе *Anlage zur Architectonic* **2** (Riga, 1771), §875; Клаузен вывел свою формулу для $a_n = 1$ в работе *Crelle* **3** (1828), 95, а Х. Ф. Шерк (H. F. Scherk) представил доказательство в *Crelle* **9** (1832), 162–163. Если $a_n = n$ и $x = \frac{1}{2}$, получим соотношение

$$\beta = \sum_{n\geq 1}\frac{n}{2^n-1} = \sum_{m\geq 1}\left(m\left(\frac{2^m+1}{2^m-1}\right) + \frac{2^m}{(2\text{m-}1)^2}\right)2^{-m^2}$$
$$= 2.74403\ 38887\ 59488\ 36048\ 02148\ 91492\ 27216\ 43114+.$$

Эта константа появляется в приближенных формулах (20), где $B'_N \sim (\beta-2)N$ и $C'_N \sim (\frac{1}{2}\beta - \frac{1}{4}\alpha - \frac{1}{2})N$.]

Кстати, если положить $q = x$ и $z = xy$ в первом тождестве из упр. 5.1.1–16, а затем вычислить $\frac{\partial}{\partial y}$ при $y = 1$, получим интересное тождество:

$$\sum_{n\geq 1}\frac{x^n}{1-x^n} = \sum_{k\geq 1} kx^k(1-x^{k+1})(1-x^{k+2})\ldots.$$

28. Потомками k-го узла являются узлы $3k-1$, $3k$ и $3k+1$; родителем является $\lfloor (k+1)/3\rfloor$. Время выполнения MIX-программы, аналогичной программе H, асимптотически приближается к $21\frac{2}{3}N\log N \approx 13.7N\lg N$ машинным циклам. Используя идею из упр. 18, можно сократить его до $18\frac{2}{3}N\log_3 N \approx 11.8N\lg N$, хотя за счет операций деления на 3 добавится большое слагаемое $\Theta(N)$.

Более подробную информацию о t-нарных деревьях можно найти в работе S. Okoma, *Lecture Notes in Comp. Sci.* **88** (1980), 439–451.

30. Предположим, что $n = 2^t - 1 + r$, где $t = \lfloor \lg n\rfloor$ и $1 \leq r \leq 2^t$. Тогда $h_{2m} = [m=0]$, и получим

$$h_{(n+1)m} \leq \sum_{j=0}^{t-2}(2^j-1)h_{n(m-j)} + 2^{t-1}h_{n(m-t+1)} + rh_{n(m-t)} \qquad \text{при } n \geq 2,$$

рассматривая число элементов на уровне j, которые претендуют стать окончательным "пристанищем" для K_{n+1} после того, как он будет "протянут" на место K_1. Таким образом, если $g_{nm} = h_{nm}/2^m$, получим

$$g_{(n+1)m} \leq \sum_{j=0}^{t-1}\frac{2^j-1}{2^j}g_{n(m-j)} + g_{n(m-t+1)} + \frac{r}{2^t}g_{n(m-t)} \leq (\lg(n+1))\max_{m\geq 0} g_{nm},$$

из чего по индукции следует $g_{nm} \leq L_n = \prod_{k=2}^{n}\lg k$.

Среднее суммарное число продвижений во время фазы выбора равно

$$B''_N = h_N^{-1} \sum_{m \geq 0} m h_{Nm} ,$$

где $h_N = \sum_{m \geq 0} h_{Nm}$ есть суммарное число возможных пирамид (теорема H). Известно, что $B''_N \leq N\lceil \lg N \rceil$. С другой стороны, имеем $B''_N \geq m - h_N^{-1} \sum_{k=1}^{m} (m-k) h_{Nk} \geq m - h_N^{-1} L_N \sum_{k=1}^{m} (m-k) 2^k > m - 2^{m+1} h_n^{-1} L_N$ для всех m. Выбор $m = \lg(h_N/L_N) + O(1)$ дает теперь $B''_N \geq \lg(h_N/L_N) + O(1)$.

Число сравнений, необходимых для формирования пирамиды, не превышает $2N$, как следует из упр. 23, (b), значит, $h_N \geq N!/2^{2N}$. Очевидно, что $L_N \leq (\lg N)^N$, и получим $\lg(h_N/L_N) \geq N \lg N - N \lg\lg N + O(N)$. [*J. Algorithms* **15** (1993), 76–100.]

31. (Решение предложено Ю. Эдигхоффер (J. Edighoffer), 1981.) Пусть A — массив, содержащий $2n$ элементов, причем $A[2\lfloor i/2 \rfloor] \leq A[2i]$ и $A[2\lfloor i/2 \rfloor - 1] \geq A[2i-1]$ при $1 < i \leq n$. Кроме того, потребуем, чтобы $A[2i-1] \geq A[2i]$ при $1 \leq i \leq n$. (В соответствии со структурой пирамид последнее условие выполняется для всех i тогда и только тогда, когда оно выполняется при $n/2 < i \leq n$.) Эти "пирамиды-близнецы" содержат $2n$ элементов; если общее число элементов нечетно, то один элемент просто хранится отдельно. Для работы с пирамидами-близнецами можно воспользоваться соответствующими модификациями алгоритмов этого раздела; интересно разработать их во всех деталях. К этой идее независимо пришли и проанализировали ее авторы работы J. van Leeuwen, D. Wood, *Comp. J.* **36** (1993), 209–216, в которой такая структура названа интервальной пирамидой.

32. В любой пирамиде из N различных элементов $m = \lceil N/2 \rceil$ наибольших элементов образуют поддерево. Не менее $\lfloor m/2 \rfloor$ из них должны не быть листьями в этом поддереве, поскольку в двоичном дереве с k листьями имеется, по меньшей мере, $k-1$ элементов, не являющихся листьями. Таким образом, не менее $\lfloor m/2 \rfloor$ из наибольших m элементов окажутся на первых $\lfloor N/2 \rfloor$ позициях в пирамиде. Эти элементы должны быть продвинуты на позиции в корне прежде, чем они достигнут своего окончательного положения. Следовательно, доля, вносимая в общее число B путем этого перемещения, составляет не менее $\sum_{k=1}^{\lfloor m/2 \rfloor} \lfloor \lg k \rfloor = \frac{1}{2} m \lg m + O(m)$, как следует из упр. 1.2.4–42. Тогда $B_{\min}(N) \geq \frac{1}{4} N \lg N + O(N) + B_{\min}(\lfloor N/2 \rfloor)$ и требуемый результат получается индукцией по N. [I. Wegener, *Theoretical Comp. Sci.* **118** (1993), 81–98, теорема 5.1. Р. Шаффер, Р. Седгевик и независимо от них Боллобаш (Bollobás), Феннер (Fenner) и Фризе (Frieze) построили перестановки, которые требуют не менее $\frac{1}{2} N \lg N + O(N \log\log N)$ продвижений; см. *J. Algorithms* **15** (1993), 76–100; **20** (1996), 205–217. Такие перестановки довольно редки, как следует из упр. 30.]

33. Пусть `P` и `Q` указывают на данные приоритетные очереди; как и в основном тексте раздела, в следующем ниже алгоритме полагается, что `DIST`$(\Lambda) = 0$, хотя Λ и не является узлом.

M1. [Начальная установка.] Установить `R` $\leftarrow \Lambda$.

M2. [Слияние списков.] Если `Q` $= \Lambda$, установить `D` $\leftarrow$ `DIST(P)` и перейти к M3. Если `P` $= \Lambda$, то установить `P` $\leftarrow$ `Q`, `D` $\leftarrow$ `DIST(P)` и перейти к M3. Иначе, если `KEY(P)` $\geq$ `KEY(Q)`, установить `T` $\leftarrow$ `RIGHT(P)`, `RIGHT(P)` $\leftarrow$ `R`, `R` $\leftarrow$ `P`, `P` $\leftarrow$ `T` и повторить шаг M2. Если `KEY(P)` $<$ `KEY(Q)`, установить `T` $\leftarrow$ `RIGHT(Q)`, `RIGHT(Q)` $\leftarrow$ `R`, `R` $\leftarrow$ `Q`, `Q` $\leftarrow$ `T` и повторить шаг M2. (Этот шаг, по сути, сливает два "правосторонних списка" данных деревьев, при этом в поля `RIGHT` временно помещаются указатели вверх.)

M3. [Выполнено?] Если `R` $= \Lambda$, завершить выполнение алгоритма; `P` указывает на ответ.

M4. [Зафиксировать поля `DIST`.] Установить `Q` $\leftarrow$ `RIGHT(R)`. Если `DIST(LEFT(R))` $<$ `D`, то установить `D` $\leftarrow$ `DIST(LEFT(R))` $+1$, `RIGHT(R)` $\leftarrow$ `LEFT(R)`, `LEFT(R)` $\leftarrow$ `P`; иначе —

установить D ← D + 1, RIGHT(R) ← P. Наконец, установить DIST(R) ← D, P ← R, R ← Q и вернуться к шагу M3. ▌

34. Начав с рекуррентного соотношения

$$L_1(z) = z, \qquad L_{m+1}(z) = L_m(z)\left(L(z) - \sum_{k=1}^{m-1} L_k(z)\right),$$

для составляющих обобщенной производящей функции $L(z) = \sum_{n\geq 0} l_n z^n = \sum_{m\geq 1} L_m(z)$, где $L_m(z) = z^{2^{m-1}} + \cdots$ порождает левосторонние деревья с кратчайшим путем длиной m от корня до Λ, Райнер Кемр (Rainer Kemp) доказал, что $L(z) = z + \frac{1}{2}L(z)^2 + \frac{1}{2}\sum_{m\geq 1} L_m(z)^2$ и что $a \approx 0.25036$ и $b \approx 2.7494879$ [см. *Inf. Proc. Letters* **25** (1987), 227–232; *Random Graphs '87* (1990), 103–130]. Луис Трабб Пардо (Luis Trabb Pardo) в 1978 году обратил внимание на то, что производящая функция $G(z) = zL(z)$ удовлетворяет очень изящному отношению $G(z) = z + G(zG(z))$.

35. Пусть поле DIST исключенного узла равно d_0, а поле DIST слитых поддеревьев равно d_1. Если $d_0 = d_1$, то подниматься вверх вообще не нужно. Если $d_0 > d_1$, то затем $d_1 = d_0 - 1$, и если подняться на n уровней, то новые поля DIST предков узла P будут равняться соответственно $d_1 + 1$, $d_1 + 2$, ..., $d_1 + n$. Если $d_0 < d_1$, восходящий путь должен вести только влево.

36. Вместо обобщенной приоритетной очереди проще всего воспользоваться двусвязным списком; при "использовании" узлов помещайте их в один конец списка, а исключайте — с другого конца. [См. анализ "самоорганизующихся" массивов в разделе 6.1.]

37. В бесконечной пирамиде самый большой k-й элемент с равной вероятностью может оказаться и в правой, и в левой подпирамидах своего большего предка. Таким образом, можно использовать теорию цифровых деревьев поиска и получить $e(k) = \overline{C}_k - \overline{C}_{k-1}$ в обозначениях выражения 6.3–(13). Как следует из упр. 6.3–28, получим $e(k) = \lg k + \gamma/(\ln 2) + \frac{1}{2} - \alpha + \delta_0(k) + O(k^{-1}) \approx \lg k - .274$, где α определено в (19), а $\delta_0(k)$ — периодическая функция $\lg k$. [P. V. Poblete, *BIT* **33** (1993), 411–412.]

38. $M_0 = \emptyset$; $M_1 = \{1\}$; $M_N = \{N\} \uplus M_{2^k-1} \uplus M_{N-2^k}$ при $N > 1$, где $k = \lfloor \lg(2N/3) \rfloor$.

РАЗДЕЛ 5.2.4

1. Начните с $i_1 = \cdots = i_k = 1$, $j = 1$. Теперь повторяйте следующие операции: найти $\min(x_{1i_1}, \ldots, x_{ki_k}) = x_{ri_r}$ и установить $z_j = x_{ri_r}$, $j \leftarrow j + 1$, $i_r \leftarrow i_r + 1$. (В этом случае будет удобно положить, что $x_{i(m_i+1)} = \infty$.)

Если k не слишком велико, то желательно хранить ключи $x_{1i_1}, \ldots, x_{ki_k}$ в виде древовидной структуры, которая позволяет осуществлять многократный выбор, как обсуждалось в разделе 5.2.3; тогда потребуется всего $\lfloor \lg k \rfloor$ сравнений, чтобы найти каждый новый минимум, кроме первого. На самом деле это типичное применение принципа "наименьший из включенных первым исключается" в приоритетной очереди. Можно хранить ключи в пирамиде и вообще не использовать ∞. Дальнейшее обсуждение приводится в разделе 5.4.1.

2. Пусть C — число сравнений; тогда $C = m + n - S$, где S — количество элементов, передаваемых на шаге M4 или M6. Как легко видеть, вероятность того, что $S \geq s$, равна

$$q_s = \left(\binom{m+n-s}{m} + \binom{m+n-s}{n}\right) \Big/ \binom{m+n}{m}$$

при $1 \leq s \leq m + n$; $q_s = 0$ при $s > m + n$. Следовательно, математическое ожидание величины S есть $\mu_{mn} = q_1 + q_2 + \cdots = m/(n+1) + n/(m+1)$ [ср. с упр. 3.4.2–5, 6], а дисперсия равна $\sigma^2_{mn} = (q_1 + 3q_2 + 5q_3 + \cdots) - \mu^2_{mn} = m(2m+n)/(n+1)(n+2) + (m+2n)n/(m+1)(m+$

$2) - \mu_{mn}^2$. Таким образом, $C = (\min \min(m,n),\ \ \text{ave}\ m+n-\mu_{mn},\ \ \max m+n-1,\ \ \text{dev}\ \sigma_{mn})$. Для случая, когда $m = n$, среднее значение первым нашел Х. Нэглер (H. Nagler) [*CACM* **3** (1960), 618–620]; асимптотическое выражение для него имеет вид $2n - 2 + O(n^{-1})$, а выражение для стандартного отклонения — $\sqrt{2} + O(n^{-1})$. Таким образом, C недалеко отклоняется от своего максимального значения.

3. **M2′.** Если $K_i < K'_j$, перейти к шагу M3′; если $K_i = K'_j$, перейти к шагу M7′; если $K_i > K'_j$, перейти к шагу M5′.

M7′. Установить $K''_k \leftarrow K'_j$, $k \leftarrow k+1$, $i \leftarrow i+1$, $j \leftarrow j+1$. Если $i > M$, перейти к шагу M4′; в противном случае, если $j > N$, перейти к шагу M6′; иначе — вернуться к шагу M2′. ▌

(Соответствующим образом изменяются и другие шаги алгоритма M. И опять исчезнет необходимость анализировать особые случаи, если в конец обоих массивов добавить искусственные ключи $K_{M+1} = K'_{N+1} = \infty$ в конце файла.)

4. Последовательность элементов, появляющихся с течением времени в фиксированном внутреннем узле дерева выбора, получается путем слияния последовательностей элементов, появляющихся в узлах — потомках этого узла. (В разделе 5.2.3 рассматривается выбор *наибольшего* элемента, но его анализ с тем же успехом можно было провести и для обратного отношения порядка.) Таким образом, операции, которые осуществляются при выборе из дерева, по существу, те же самые, что и при слиянии, но выполняются они в другом порядке и с использованием других структур данных.

Другие общие черты слияния и выбора из дерева рассмотрены в упр. 1. Заметим, что N-путевое слияние одноэлементных массивов — это сортировка посредством выбора; сравните также четырехпутевое слияние (A, B, C, D) с двухпутевым слиянием (A, B), (C, D), а затем с (AB, CD).

5. На шаге N6 всегда $K_i < K_{i-1} \le K_j$; на шаге N10 всегда $K_j < K_{j+1} < K_i$.

6. 2 6 4 10 8 14 12 16 15 11 13 7 9 3 5 1. После первого просмотра имеем 1 2 5 6 7 8 13 14 16 15 12 11 10 9 4 3 (две из предполагаемых четырех ступенек вниз исчезли). Эту возможность заметил Д. А. Белл (D. A. Bell) [см. *Comp. J.* **1** (1958), 74]. Из-за наличия не совсем предсказуемых ситуаций наподобие этой попытки точно проанализировать алгоритм N почти безнадежны.

7. $\lceil \lg N \rceil$, если $N > 1$. (Задумайтесь над тем, сколько раз нужно удвоить значение p, прежде чем оно станет $\ge N$.)

8. Если N не кратно $2p$, то при таком просмотре встретится одна более короткая серия и она всегда будет находиться где-то в середине. Пусть ее длина равна t, $0 \le t < p$. На шаге S12 обрабатывается ситуация, когда короткая серия должна быть "слита" с пустой серией, т. е. $t = 0$; в противном случае мы, по существу, получим $x_1 \le x_2 \le \cdots \le x_p \mid y_t \ge \cdots \ge y_1$. Если $x_p \le y_t$, то левая серия будет обработана первой и от шага S6 после пересылки x_p мы перейдем к шагу S13. С другой стороны, если $x_p > y_t$, то по отношению к правому подмассиву будет искусственно имитировано завершение обработки, но $K_j = x_p$ никогда не будет $< K_i$ на шаге S3! Таким образом, во всех случаях из шага S6 мы, в конце концов, попадем на шаг S13.

10. Например, в алгоритме M можно сливать элементы $x_{j+1} \ldots x_{j+m}$ с $x_{j+m+1} \ldots x_{j+m+n}$ и помещать результат в позиции $x_1 \ldots x_{m+n}$ массива, не создавая при этом никаких конфликтных ситуаций, если только $j \ge n$. Приложив некоторые усилия, эту идею можно развить настолько, что для всей сортировки потребуется $N + 2^{\lfloor \lg N \rfloor - 1}$ ячеек. Но по сравнению с алгоритмом S такая программа кажется довольно сложной. [См. *Comp. J.* **1** (1958), 75; см. также Л. С. Лозинский, *Кибернетика* **1**, 3 (1965), 58–62.]

11. Да. Это можно показать, например, рассмотрев родство с методом выбора из дерева, упомянутое в упр. 4. Но алгоритмы N и S, очевидно, неустойчивы.

12. Установить $L_0 \leftarrow 1$, $t \leftarrow N+1$; затем при $p = 1, 2, \ldots, N-1$ выполнить следующие действия.

Если $K_p \le K_{p+1}$, установить $L_p \leftarrow p+1$; в противном случае установить $L_t \leftarrow -(p+1)$, $t \leftarrow p$.

И наконец, установить $L_t \leftarrow 0$, $L_N \leftarrow 0$, $L_{N+1} \leftarrow |L_{N+1}|$.

(Устойчивость сохраняется. Число просмотров равно $\lceil \lg r \rceil$, где r — число восходящих серий в начальном массиве. Точное распределение величины r проанализировано в разделе 5.1.3. Можно сделать вывод о том, что при использовании связного распределения памяти "естественное" слияние предпочтительнее "простого", хотя при последовательном распределении наблюдалась обратная ситуация.)

13. При $N \ge 3$ время выполнения программы равно $(11A+6B+3B'+9C+2C''+4D+5N+9)u$, где A — число просмотров, $B = B' + B''$ — число выполненных операций слияния подмассивов, B' — число таких слияний, в которых p-подмассив был обработан первым, $C = C' + C''$ — число выполненных сравнений, C' — число сравнений с результатом $K_p \le K_q$, $D = D' + D''$ — число элементов, остававшихся в подмассивах, после того как один подмассив был исчерпан, D' — число таких элементов, принадлежащих q-подмассиву. Для табл. 3 имеем $A = 4$, $B' = 6$, $B'' = 9$, $C' = 22$, $C'' = 22$, $D' = 10$, $D'' = 10$; суммарное время $= 761u$. (Конкурирующая программа 5.2.1L требует только $433u$, если внести изменения, описанные в упр. 5.2.1–33. В результате приходим к выводу, что слияние не особенно эффективно при малых N.)

Алгоритм L выполняет ряд операций слияния подмассивов, размеры которых (m, n) можно определить следующим образом. Пусть в двоичной системе счисления $N - 1 = (b_k \ldots b_1 b_0)_2$. Выполняется $(b_k \ldots b_{j+1})_2$ "обычных" слияний, таких, что $(m, n) = (2^j, 2^j)$ при $0 \le j < k$. Имеются также "особые" слияния, такие, что $(m, n) = (2^j, 1 + (b_{j-1} \ldots b_0)_2)$, если только $b_j = 1$, при $0 \le j \le k$. Например, при $N = 14$ выполняется шесть обычных слияний $(1, 1)$, три обычных слияния $(2, 2)$, одно обычное слияние $(4, 4)$; особые слияния выполняются с подмассивами размером $(1, 1)$, $(4, 2)$, $(8, 6)$. Мультимножество M_N размеров слияний (m, n) можно также описать рекуррентным соотношением

$$M_1 = \emptyset; \qquad M_{2^k+r} = \{(2^k, r)\} \uplus M_{2^k} \uplus M_r \quad \text{при } 0 < r \le 2^k.$$

Отсюда заключаем, что независимо от распределения, характеризующего исходный массив, $A = \lceil \lg N \rceil$, $B = N - 1$, $C' + D'' = \sum_{j=0}^{k} b_j 2^j (1 + \frac{1}{2} j)$, $C'' + D' = \sum_{j=0}^{k} b_j (1 + 2^j (\frac{1}{2} j + b_{j+1} + \cdots + b_k))$. Следовательно, только параметры B', C', D' нуждаются в дальнейших исследованиях.

Если исходный массив случаен, то каждая операция слияния удовлетворяет условиям упр. 2 и не зависит от выполнения других аналогичных операций слияния, так что распределения параметров B', C', D' являются "свертками" их распределений для каждого отдельного слияния подмассивов. Средние значения для такой операции равны $B' = n/(m+n)$, $C' = mn/(n+1)$, $D' = n/(m+1)$. Чтобы получить точные средние значения, достаточно просуммировать эти величины по всевозможным подходящим парам (m, n).

Если $N = 2^k$, то имеет место простейшая ситуация: $B'_{\text{ave}} = \frac{1}{2} B$, $C'_{\text{ave}} = \frac{1}{2} C_{\text{ave}}$, $C + D = kN$ и $D_{\text{ave}} = \sum_{j=1}^{k} (2^{k-j} 2^j / (2^{j-1} + 1)) = \alpha' N + O(1)$, где значение

$$\alpha' = \sum_{n \ge 0} \frac{1}{2^n + 1} = \alpha + \frac{1}{2} - 2 \sum_{n \ge 1} \frac{1}{4^n - 1}$$

$$= 1.26449\ 97803\ 48444\ 20919\ 13197\ 47255\ 49848\ 25577-$$

можно вычислить с высокой точностью, как в упр. 5.2.3–27. Этот частный случай проанализировали Э. Глисон (A. Gleason) [неопубликовано, 1956] и Х. Нэглер (H. Nagler) [*CACM* **3** (1960), 618–620].

14. Чтобы получить максимальное значение параметра C, достаточно в упр. 13 положить $D = B$. [Детальный анализ алгоритма L выполнен в работе W. Panny, H. Prodinger, *Algorithmica* **14** (1995), 340–354.]

15. Продублируйте команды реализации шагов L3, L4 и L6 для случаев, когда известно, что L_s равно p или q. [Алгоритм можно *еще более* усовершенствовать, удалив из внутреннего цикла присвоение $s \leftarrow p$ (или $s \leftarrow q$), просто переименовав регистры! Замените, например, строки 20 и 21 строкой `LD3 INPUT,1(L)` и продолжайте далее, считая, что p находится в rI3, s — в rI1, а значение переменной L_s, L, заведомо равно p. Имея двенадцать копий внутреннего цикла, соответствующих различным перестановкам (p, q, r) относительно регистров (rI1, rI2, rI3) и различной информации о L_s, можно сократить среднее время работы до $8N \lg N + O(N)$.]

16. Полученный алгоритм будет работать чуть быстрее алгоритма L (см. упр. 5.2.3–28).

17. Рассматривайте новую запись как подмассив длиной 1. Повторяйте слияние двух наименьших подмассивов, имеющих одинаковую длину. (Полученный алгоритм сортировки, по существу, такой же, как алгоритм L, только слияния подмассивов выполняются в другом порядке.)

18. Да, но это, по-видимому, очень сложно. В простейшем известном методе используется следующее искусное построение [*ДАН СССР* **186** (1969), 1256–1258]. Пусть $n \approx \sqrt{N}$. Разделим массив на $m+2$ "зон" $Z_1 \ldots Z_m\, Z_{m+1}\, Z_{m+2}$, где зона Z_{m+2} содержит $N \bmod n$ записей, а остальные зоны содержат ровно по n записей. Поменяем местами записи зоны Z_{m+1} с зоной, в которой содержится R_M; теперь массив имеет вид $Z_1 \ldots Z_m\, A$, где каждая из зон $Z_1 \ldots Z_m$ содержит ровно n упорядоченных записей, а A — вспомогательная область, содержащая s записей, где s — некоторое число из диапазона $n \le s < 2n$.

Найдем зону с наименьшим лидирующим элементом и поменяем ее местами с Z_1. Если более одной зоны имеет наименьший лидирующий элемент, выберите ту из них, замыкающий элемент которой будет наименьшим. (Для этого потребуется $O(m+n)$ операций.) Затем найдем зону со следующим по порядку лидирующим элементом и поменяем ее местами с Z_2 и т. д. В конце концов, за $O(m(m+n)) = O(N)$ операций мы перекомпонуем эти m зон таким образом, чтобы их лидирующие элементы были упорядочены. Кроме того, в соответствии с исходным предположением о характере массива каждый из ключей в зонах $Z_1 \ldots Z_m$ теперь имеет не более чем n инверсий.

Для слияния Z_1 с Z_2 можно воспользоваться следующим трюком: поменять местами Z_1 с первыми n элементами A' области A, а затем слить Z_2 с A', как обычно, но при выводе поменять местами элементы с элементами области $Z_1 Z_2$. Например, если $n = 3$ и $x_1 < y_1 < x_2 < y_2 < x_3 < y_3$, то имеем следующее.

	Зона 1			Зона 2			Вспомогательная область		
Начальное содержимое:	x_1	x_2	x_3	y_1	y_2	y_3	a_1	a_2	a_3
Обмен с Z_1:	a_1	a_2	a_3	y_1	y_2	y_3	x_1	x_2	x_3
Обмен с x_1:	x_1	a_2	a_3	y_1	y_2	y_3	a_1	x_2	x_3
Обмен с y_1:	x_1	y_1	a_3	a_2	y_2	y_3	a_1	x_2	x_3
Обмен с x_2:	x_1	y_1	x_2	a_2	y_2	y_3	a_1	a_3	x_3
Обмен с y_2:	x_1	y_1	x_2	y_2	a_2	y_3	a_1	a_3	x_3
Обмен с x_3:	x_1	y_1	x_2	y_2	x_3	y_3	a_1	a_3	a_2

(Слияние всегда завершается в тот момент, когда происходит обмен n-го элемента вспомогательной области. В результате применения этого метода записи вспомогательной области обычно переставляются.)

Тот же трюк применяется при слиянии сначала зон Z_1 и Z_2, затем — Z_2 и Z_3, ..., Z_{m-1} и Z_m; для этого необходимо выполнить $O(mn) = O(N)$ операций. Поскольку ни один элемент не имел более n инверсий, часть массива $Z_1 \ldots Z_m$ оказывается, таким образом, полностью рассортированной.

Для окончательной "доводки" рассортируем записи $R_{N+1-2s} \ldots R_N$ методом вставок за $O(s^2) = O(N)$ шагов. В результате s наибольших элементов окажутся в области A. Затем применим все тот же трюк с областью A и сольем $R_1 \ldots R_{N-2s}$ с $R_{N+1-2s} \ldots R_{N-s}$ (но поменяем везде ролями "левое" и "правое", а также отношения "меньше" и "больше"). В конце рассортируем записи $R_{N+1-s} \ldots R_N$ методом вставок.

Последующее усовершенствование рассматривается в работе J. Katajainen, T. Pasanen, J. Teuhola, *Nordic J. Computing* **3** (1996), 27–40. Проблема *устойчивости* слияния на месте рассматривается в ответе к упр. 5.5–3.

19. Вагоны можно перенумеровать таким образом, чтобы в конечной перестановке их номера были упорядочены: 1 2 ... 2^n; так что это, по существу, задача сортировки. Прогоним сначала первые 2^{n-1} вагонов через $n-1$ тупиков, расположив их в порядке убывания номеров и поместив их в n-й тупик так, чтобы наименьший элемент оказался на входе в тупик (на вершине стека). Затем прогоним остальные 2^{n-1} вагонов через $n-1$ тупиков, расположив их в порядке возрастания непосредственно перед n-м тупиком. И наконец, сольем эти две последовательности очевидным способом.

20. Дополнительную информацию можно найти в работе R. E. Tarjan, *JACM* **19** (1972), 341–346.

22. См. *Information Processing Letters* **2** (1973), 127–128.

23. Слияние можно представить в виде бинарного дерева, все внешние узлы которого находятся на уровнях $\lfloor \lg N \rfloor$ и $\lceil \lg N \rceil$. Таким образом, максимальное число сравнений равно длине минимального внешнего пути на бинарном дереве с N внешними узлами (формула 5.3.1–(34)) минус $N-1$, поскольку при $f(m,n) = m+n-1$ получим максимум и имеется $N-1$ слияний (см. также соотношение 5.4.9–(1)).

Обобщенная методика анализа асимптотических свойств такого рекуррентного соотношения при помощи преобразования Меллина представлена в работе P. Flajolet, M. Golin, *Acta Informatica* **31** (1994), 673–696. В частности, в ней показано, что среднее количество сравнений равно $N \lg N - \theta N + \delta(\lg N)N + O(1)$, а дисперсия $\approx .345N$. Здесь δ есть непрерывная функция с периодом 1 и средним значением 0, а

$$\theta = \frac{1}{\ln 2} - \frac{1}{2} + \frac{1}{\ln 2} \sum_{m=1}^{\infty} \frac{2}{(m+1)(m+2)} \ln \frac{2m+1}{2m}$$
$$= 1.24815\,20420\,99653\,84890\,29565\,64329\,53240\,16127+.$$

Суммарное число сравнений при $N \to \infty$ хорошо аппроксимируется методом нормального распределения; дополнительный анализ выполнен в работе H.-K. Hwang, M. Cramer, *Random Structures и Algorithms* **8** (1996), 319–336; **11** (1997), 81–96.

РАЗДЕЛ 5.2.5

1. Нет, потому что после первого просмотра поразрядная сортировка вообще не будет выполняться, если распределяющая сортировка неустойчива. (Но предложенный способ распределяющей сортировки *можно было бы* применить к поразрядной СЦ-сортировке, на что указано в последнем абзаце этого раздела.)

2. Это алгоритм "антиустойчивой" сортировки, прямо противоположной устойчивой. Элементы с одинаковыми ключами располагаются в обратном порядке, поскольку при первом просмотре записи сканируются от R_N до R_1. (Это действительно удобно, так как в строках 28 и 20 программы R Λ приравнивается к 0, но, разумеется, вовсе необязательно выполнять первый просмотр в обратном направлении.)

3. Если стопка 0 не пуста, то `BOTM[0]` уже указывает на первую запись; если же она пуста, то мы устанавливаем `P` $\leftarrow$ `LOC(BOTM[0])`, а впоследствии устанавливаем в `LINK(P)` указатель на первую непустую стопку.

4. Если осталось выполнить четное число просмотров, то поместить стопку 0 в начало стека (в очередности от первых элементов к последним), затем стопку 1, ..., стопку $(M-1)$; в результате записи будут упорядочены относительно уже проанализированных разрядов ключей. Если осталось выполнить нечетное число просмотров, то поместите в начало стека стопку $(M-1)$, затем — стопку $(M-2)$, ..., стопку 0; в результате записи расположатся в обратном порядке относительно уже рассмотренных разрядов ключей. (Это правило, по-видимому, впервые опубликовал Э. Х. Френд (E. H. Friend) [*JACM* **3** (1956), 156, 165–166].)

5. Замените строку 04 строкой `ENT3 7`, а таблицы `R3SW` и `R5SW` следующими таблицами.

```
R3SW    LD2   KEY,1(1:1)
        LD2   KEY,1(2:2)
        LD2   KEY,1(3:3)
        LD2   KEY,1(4:4)
        LD2   KEY,1(5:5)
        LD2   INPUT,1(1:1)
        LD2   INPUT,1(2:2)
        LD2   INPUT,1(3:3)
R5SW    LD1   INPUT,1(LINK)
        .
        .     (Повторить предыдущую строку еще шесть раз.)
        DEC1  1                ▮
```

Время выполнения новой программы вычисляется путем замены "3" на "8"; оно равняется $(11p-1)N+16pM+12p-4E+2$ при $p=8$.

6. (a). Проанализируйте размещение $(N+1)$-го элемента. Рекуррентное соотношение

$$p_{M(N+1)k} = \frac{k+1}{M}\, p_{MN(k+1)} + \frac{M-k}{M}\, p_{MNk}$$

эквивалентно указанной в условии формуле. (b) n-я производная удовлетворяет соотношению $g^{(n)}_{M(N+1)}(z) = (1-n/M)g^{(n)}_{MN}(z) + ((1-z)/M)g^{(n+1)}_{MN}(z)$, что доказывается индукцией по n. Полагая, что $z=1$, находим $g^{(n)}_{MN}(1) = (1-n/M)^N M^{\underline{n}}$, поскольку $g_{M0}(z) = z^M$. Следовательно, $\mathrm{mean}(g_{MN}) = (1-1/M)^N M$, $\mathrm{var}(g_{MN}) = (1-2/M)^N M(M-1) + (1-1/M)^N M - (1-1/M)^{2N} M^2$. (Обратите внимание на то, что производящая функция для E в программе R есть $g_{MN}(z)^p$.)

7. Пусть R — поразрядная сортировка, RX — обменная поразрядная сортировка. Укажем некоторые важные сходные черты и отличия. RX продвигается от старших цифр к младшим, R — наоборот. В обоих методах просматриваются цифры ключей, а сами ключи не сравниваются. В RX всегда $M=2$ (см., однако, упр. 1). Время выполнения для R почти не меняется, а для RX оно очень чувствительно к распределению цифр. В обоих случаях время выполнения пропорционально $O(N\log K)$, где K — диапазон изменения ключей, но константа пропорциональности для RX больше. С другой стороны, если цифры старших разрядов ключей распределены равномерно, то среднее время выполнения для RX будет

равно $O(N \log N)$ независимо от величины K. Сортировка R требует наличия полей связи, а RX работает с "минимальной памятью". Внутренний цикл в R больше подходит для компьютеров с конвейерной архитектурой.

8. При последнем просмотре стопки следует сцепить в другом порядке; например, если $M = 256$, то стопка $(10000000)_2$ должна быть первой, за ней — стопка $(10000001)_2$, ..., стопка $(11111111)_2$, стопка $(00000000)_2$, стопка $(00000001)_2$, ..., стопка $(01111111)_2$. Это изменение порядка сцепления легко осуществить путем модификации алгоритма H или изменения стратегии распределения памяти при последнем просмотре (см. табл. 1).

9. Можно сначала отделить отрицательные ключи от положительных, как в упр. 5.2.2–33, или же при первом просмотре преобразовать ключи и записать их в дополнительном коде. Можно также после последнего просмотра отделить положительные ключи от отрицательных, изменив порядок расположения последних, но метод из упр. 5.2.2–33 для этого уже не годится.

11. Без первого просмотра ключи при помощи нашего метода все равно будут полностью рассортированы, потому что (по чистой случайности) ключ 503 уже предшествует ключу 509. Без первых двух просмотров было бы $1 + 1 + 0 + 0 + 0 + 1 + 1 + 1 + 0 + 0 = 5$ инверсий.

12. После обмена R_k с R[P] на шаге M4 (упр. 5.2–12) можно сравнить K_k с K_{k-1}. Если K_k меньше, то сравниваем его с K_{k-2}, K_{k-3}, ..., пока не встретим $K_k \geq K_j$. Тогда устанавливаем $(R_{j+1}, \ldots, R_{k-1}, R_k) \leftarrow (R_k, R_{j+1}, \ldots, R_{k-1})$, *не меняя* полей LINK. Удобно принудительно поместить с левого конца массива искусственный ключ K_0, который $\leq$ всех других ключей.

14. Если исходная перестановка карт требует k чтений в смысле упр. 5.1.3–20 и если при каждом просмотре карты раскладываются в m стопок, то придется выполнить не менее $\lceil \log_m k \rceil$ просмотров. (Рассмотрите обратный переход — от упорядоченной колоды к исходной; при каждом просмотре число чтений увеличивается не более чем в m раз.) Данная перестановка требует четырех чтений для возрастающего порядка и десяти чтений для убывающего порядка. Поэтому сортировка в порядке убывания требует четырех просмотров при двух стопках и трех просмотров при трех стопках.

И обратно, этого оптимального числа просмотров достаточно для сортировки. Присвойте картам номера от 0 до $k - 1$ в соответствии с тем, на каком по порядку сеансе была прочитана карта, и примените поразрядную МЦ-сортировку в системе счисления с основанием m. [См. *Martin Gardner's Sixth Book of Mathematical Games* (San Francisco: W. H. Freeman, 1971), 111–112.]

15. Пусть требуется k чтений и можно раскладывать карты в m стопок. Порядок обращается при каждом просмотре: если необходимо k чтений в одном направлении, то в обратном направлении понадобится $n + 1 - k$ чтений. Минимальное число просмотров есть либо наименьшее четное число, большее или равное $\log_m k$, либо наименьшее нечетное число, большее или равное $\log_m(n + 1 - k)$. (Если отталкиваться от рассортированной перестановки, то нетрудно заметить, что после первого просмотра перестановка требует не более чем m чтений для убывающего порядка, после второго — не более чем m^2 чтений для возрастающего порядка и т. д.) Нашу перестановку можно рассортировать в порядке возрастания за $\min(2, 5) = 2$ просмотра, а в порядке убывания — за $\min(3, 4) = 3$ просмотра, если раскладывать карты только в две стопки.

16. Обеспечим, чтобы каждая строка завершалась специальным символом *null*, который меньше кода любой буквы в алфавите. Выполните поразрядную сортировку слева направо, причем перед этим нужно связать все строки в единый блок данных. Затем обработайте каждый блок ($k = 1, 2, \ldots$), в котором содержится более одной отличной

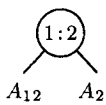
строки, разделив его на подблоки и приняв за основу k-ю букву каждой строки. При этом сохранится сортировка блоков, выполненная по начальным буквам (будем называть их *префиксом*). Когда в блоке останется только один элемент и все их k-е символы будут равны *null* (т. е. все ключи будут одинаковыми), организуем работу так, чтобы избежать его повторного анализа. [R. Paige, R. E. Tarjan, *SICOMP* **16** (1987), 973–989, §2.] Этот процесс, по сути, есть построение дерева, как описано в разделе 6.3. Более простой, но и немного менее эффективный алгоритм базируется на поразрядной сортировке справа налево и описывается в работе Aho, Hopcroft, Ullman, *The Design and Analysis of Computer Algorithms* (Addison-Wesley, 1974), 79–84. Метод Мак-Илроя, на который имеется ссылка в тексте раздела, обеспечивает на практике еще более быструю сортировку.

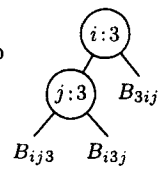
17. Метод Мак-Ларена позволяет ускорить выполнение на втором уровне, но не может использоваться на первом уровне, поскольку он не предусматривает вычисление значений N_k.

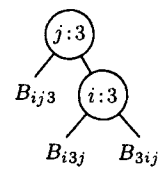
18. Во-первых докажем то, что рекомендовано в указании. Пусть $p_k = \int_{k/CN}^{(k+1)/CN} f(x)\,dx$ — вероятность того, что ключ попадет в стопку k, если имеется CN стопок. Время, необходимое для распределения записей, имеет порядок $O(N)$, а среднее число инверсий, оставшихся после распределения, составляет $\frac{1}{2}\sum_{k=0}^{CN-1}\binom{N}{j}p_k^j(1-p_k)^{N-j}\binom{j}{2} = \frac{1}{2}\sum_{k=0}^{CN-1}\binom{N}{2}p_k^2 \le \frac{N-1}{4}p_kB/C$, поскольку $p_k \le B/CN$.

Теперь рассмотрим два уровня распределения, причем на верхнем уровне число стопок — cN, и пусть $b_k = \sup\{f(x) \mid k/cN \le x < (k+1)/cN\}$. Тогда среднее суммарное время выполнения равно $O(N)$ плюс $\sum_{k=0}^{cN-1} T_k$, где T_k — среднее время, затрачиваемое при выполнении сортировки N_k ключей, для которых функция плотности вероятностей имеет вид $f_k(x) = f((k+x)/cN)/cNp_k$ в соответствии с алгоритмом Мак-Ларена. Как следует из приведенного выше анализа, $T_k = \mathrm{E}\,O(b_kN_k/cNp_k)$, поскольку $f_k(x)$ ограничена величиной b_k/cNp_k. Но $\mathrm{E}\,N_k = Np_k$, так что $T_k = O(b_k/c)$. А при $N \to \infty$ по определению интегрируемости по Риману получим $\sum_{k=0}^{cN-1} b_k \to N\int_0^1 f(x)\,dx = N$.

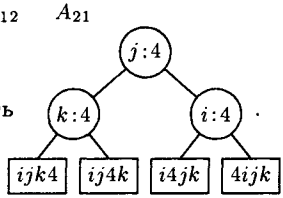
РАЗДЕЛ 5.3.1

1. (a) 

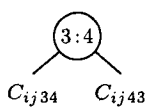
, где A_{ij} есть либо 

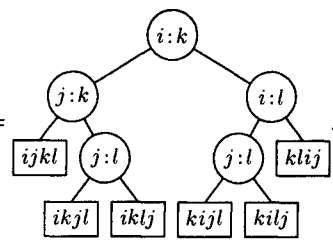
, либо 

,

а B_{ijk} есть 

. Длина внешнего пути равна 112 (оптимальная).

(b) Здесь $A_{ij} =$ 

, где $C_{ijkl} =$ 

.

Длина внешнего пути также равна 112 (оптимальная).

2. В обозначениях упр. 5.2.4–14

$$\begin{aligned} L(n) - B(n) &= \sum_{k=1}^{t}((e_k + k - 1)2^{e_k} - (e_1 + 1)2^{e_k}) + 2^{e_1+1} - 2^{e_t} \\ &= 2^{e_1} - 2^{e_t} - \sum_{k=2}^{t}(e_1 - e_k + 2 - k)2^{e_k} \\ &\geq 2^{e_1} - (2^{e_1-1} + \cdots + 2^{e_1-t+1} + 2^{e_t}) \geq 0. \end{aligned}$$

При этом равенство достигается тогда и только тогда, когда $n = 2^k - 2^j$ при некоторых $k > j \geq 0$. [Если для слияния используется "нисходящая" версия, как в упр. 5.2.4–23, максимальное число сравнений будет равно $B(n)$.]

3. При $n > 0$ число исходов, в которых наименьший ключ встречается точно k раз, равно $\binom{n}{k}P_{n-k}$. Таким образом, $2P_n = \sum_k \binom{n}{k}P_{n-k}$ при $n > 0$ и мы имеем $2P(z) = e^z P(z) + 1$, как следует из формулы 1.2.9–(10).

Другое доказательство опирается на тот факт, что $P_n = \sum_{k\geq 0} \left\{ {n \atop k} \right\} k!$, поскольку $\left\{ {n \atop k} \right\}$ есть число способов разбиения множества из n элементов на k непустых подмножеств, а эти подмножества можно переставить $k!$ способами. Таким образом, согласно формуле 1.2.9–(23) $\sum_{n\geq 0} P_n z^n/n! = \sum_{k\geq 0}(e^z - 1)^k = 1/(2 - e^z)$.

И все же остается *еще одно*, пожалуй, наиболее интересное доказательство. Скомпонуем из всех элементов устойчивую последовательность, так что K_i предшествует K_j тогда и только тогда, когда $K_i < K_j$ или ($K_i = K_j$ и $i < j$). Среди всех возможных P_n исходов данное расположение $K_{a_1} \ldots K_{a_n}$ встретится ровно 2^k раз, где k — число восходящих серий перестановки $a_1 \ldots a_n$; следовательно, P_n можно выразить через числа Эйлера: $P_n = \sum_k \left\langle {n \atop k} \right\rangle 2^k$. Искомый результат можно получить, используя формулу 5.1.3–(20), при $z = 2$.

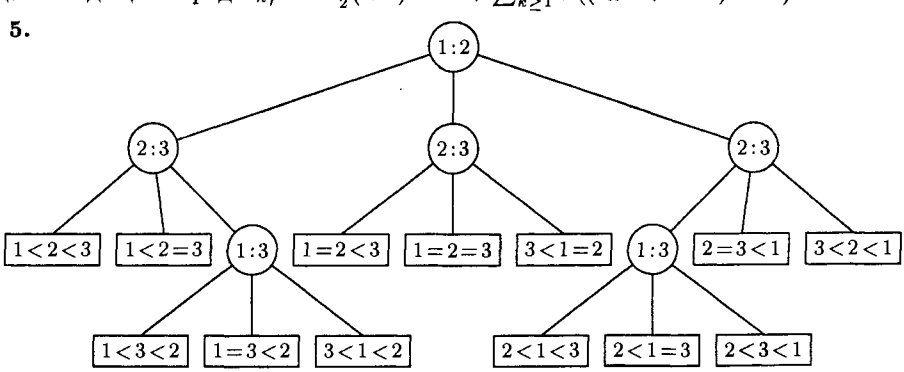
Эту производящую функцию нашел А. Кейлей (A. Cayley) [*Phil. Mag.* **18** (1859), 374–378] применительно к перечислению одного нечетко определенного класса деревьев. См. также P. A. MacMahon, *Proc. London Math. Soc.* **22** (1891), 341–344; J. Touchard, *Ann. Soc. Sci. Bruxelles* **53** (1933), 21–31; O. A. Gross, *AMM* **69** (1962), 4–8. В последней работе получена интересная формула $P_n = \sum_{k\geq 1} k^n/2^{1+k}$, $n \geq 1$.

4. Представление

$$2P(z) = \frac{1}{2}\left(1 - i\cot\frac{i(z - \ln 2)}{2}\right) = \frac{1}{2} - \frac{1}{z - \ln 2} - \sum_{k\geq 1}\left(\frac{1}{z - \ln 2 - 2\pi ik} + \frac{1}{z - \ln 2 + 2\pi ik}\right)$$

дает сходящийся ряд $P_n/n! = \frac{1}{2}(\ln 2)^{-n-1} + \sum_{k\geq 1} \Re((\ll 2 + 2\pi ik)^{-n-1})$.

5.



6. $S'(n) \geq S(n)$, поскольку все ключи могут быть различны; таким образом, остается показать, что $S'(n) \leq S(n)$. Пусть дан алгоритм сортировки для различных ключей,

требующий $S(n)$ шагов. Тогда можно построить алгоритм сортировки для общего случая, считая, что ветвь для отношения "=" идентична ветви для отношения "<", и удалить лишние сравнения. По достижении внешнего узла нам становятся известными все отношения равенства, поскольку $K_{a_1} \le K_{a_2} \le \cdots \le K_{a_n}$ и при всех $1 \le i < n$ были выполнены явные сравнения $K_{a_i} : K_{a_{i+1}}$.

М. Патерсон (M. Paterson) обратил внимание на то, что если множество ключей есть $(n_1, \ldots, n_m)$, то число сравнений может быть уменьшено до $n \lg n - \sum n_j \lg n_j + O(n)$; см. *SICOMP* **5** (1976), 2. Этой нижней границы можно достичь без существенного увеличения объема дополнительной памяти, модифицировав метод пирамидальной сортировки применительно к случаю равных ключей, как предложено в работе Munro, Raman, *Lecture Notes in Comp. Sci.* **519** (1991), 473–480.

7. См. рис. A–1. Среднее число сравнений равно $(2+3+3+2+3+3+3+6+3+3+3+2+3+3+2)/16 = 2\frac{3}{4}$.

8. См. рис. A–2. Среднее число сравнений равно $3\frac{56}{81}$.

9. Если все ключи равны, то для того, чтобы обнаружить это, необходимо выполнить, по крайней мере, $n-1$ сравнений. Обратно, $n-1$ сравнений всегда достаточно, потому что окончательное упорядочение можно получить, сравнив K_1 со всеми остальными ключами.

10. Пусть $f(n)$ — искомая функция, а $g(n)$ — минимальное среднее число сравнений, необходимое для сортировки $n+k$ элементов, где $k > 0$, и ровно k элементов имеют известные значения (0 или 1). Тогда $f(0) = f(1) = g(0) = 0$, $g(1) = 1$; $f(n) = 1 + \frac{1}{2}f(n-1) + \frac{1}{2}g(n-2)$, $g(n) = 1 + \min(g(n-1), \frac{1}{2}g(n-1) + \frac{1}{2}g(n-2)) = 1 + \frac{1}{2}g(n-1) + \frac{1}{2}g(n-2)$ при $n \ge 2$. (Таким образом, наилучшая стратегия заключается в том, чтобы сравнивать каждый раз по возможности два неизвестных ключа.) Отсюда вытекает, что $f(n) - g(n) = \frac{1}{2}(f(n-1) - g(n-1))$ при $n \ge 2$ и $g(n) = \frac{2}{3}(n + \frac{1}{3}(1 - (-\frac{1}{2})^n))$ при $n \ge 0$. Следовательно, ответ есть $\frac{2}{3}n + \frac{2}{9} - \frac{2}{9}(-\frac{1}{2})^n - (\frac{1}{2})^{n-1}$ при $n \ge 1$. (Эту точную формулу можно сравнить с теоретико-информационной оценкой нижней границы $\log_3(2^n - 1) \approx 0.6309n$.)

11. То, что $S_m(n) \le B(m) + (n-m)\lceil \lg(m+1) \rceil$ при $n \ge m$, можно доказать, используя методику бинарных вставок. С другой стороны, $S_m(n) \ge \lceil \lg \sum_{k=1}^{m} \left\{ {n \atop k} \right\} k! \rceil$, а предел правой части равен $n \lg m + O(((m-1)/m)^n)$ (см. формулу 1.2.6–(53)).

12. (a) Если избыточные сравнения не выполняются, то равным ключам, когда они сравниваются впервые, можно приписать произвольное отношение порядка, поскольку это отношение порядка нельзя вывести из предыдущих сравнений. (b) Предположим, что дерево сильно сортирует любую последовательность нулей и единиц. Докажем, что оно сильно сортирует любую перестановку элементов $\{1, 2, \ldots, n\}$. Пусть это не так, т. е. существует перестановка, которую оно якобы упорядочивает, как $K_{a_1} \le K_{a_2} \le \cdots \le K_{a_n}$, но в действительности существует i, при котором $K_{a_i} > K_{a_{i+1}}$. Заменим нулями все элементы $< K_{a_i}$ и единицами — все элементы $\ge K_{a_i}$; по предположению наш метод рассортирует полученную перестановку, если выбрать путь, ведущий к $K_{a_1} \le K_{a_2} \le \cdots \le K_{a_n}$. Значит, мы получили противоречие.

13. Если n четно, то $F(n) - F(n-1) = 1 + F(\lfloor n/2 \rfloor) - F(\lfloor n/2 \rfloor - 1)$ и нам нужно доказать, что $w_{k-1} < \lfloor n/2 \rfloor \le w_k$; это очевидно, поскольку $w_{k-1} = \lfloor w_k/2 \rfloor$. Если n нечетно, то $F(n) - F(n-1) = G(\lceil n/2 \rceil) - G(\lfloor n/2 \rfloor)$ и нам нужно доказать, что $t_{k-1} < \lceil n/2 \rceil \le t_k$; это очевидно, поскольку $t_{k-1} = \lceil w_k/2 \rceil$.

14. Согласно упр. 1.2.4–42 эта сумма равна $n\lceil \lg \frac{3}{4}n \rceil - (w_1 + \cdots + w_j)$, где $w_j < n \le w_{j+1}$. Последняя сумма равна $w_{j+1} - \lfloor j/2 \rfloor - 1$. Следовательно, $F(n)$ можно записать в виде $n\lceil \lg \frac{3}{4}n \rceil - \lfloor 2^{\lfloor \lg(6n) \rfloor}/3 \rfloor + \lfloor \frac{1}{2} \lg(6n) \rfloor$ (а также многими другими способами.)

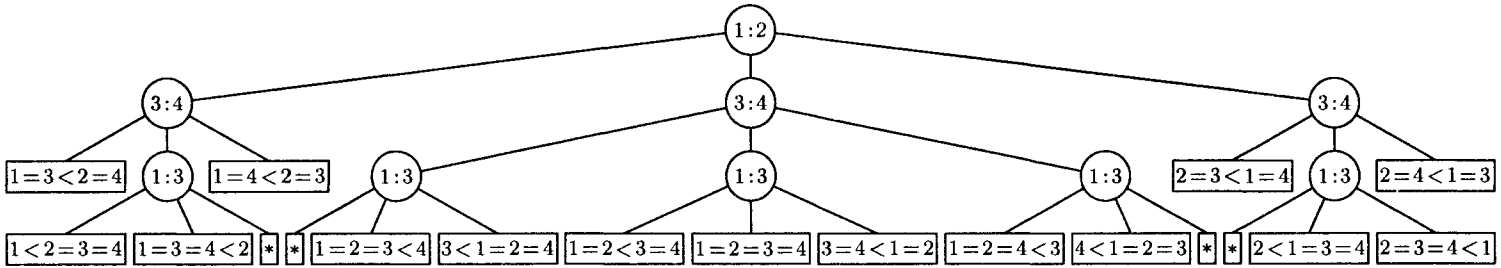


Рис. А–1. Решение упр. 7. ("*" означает невозможный случай.)

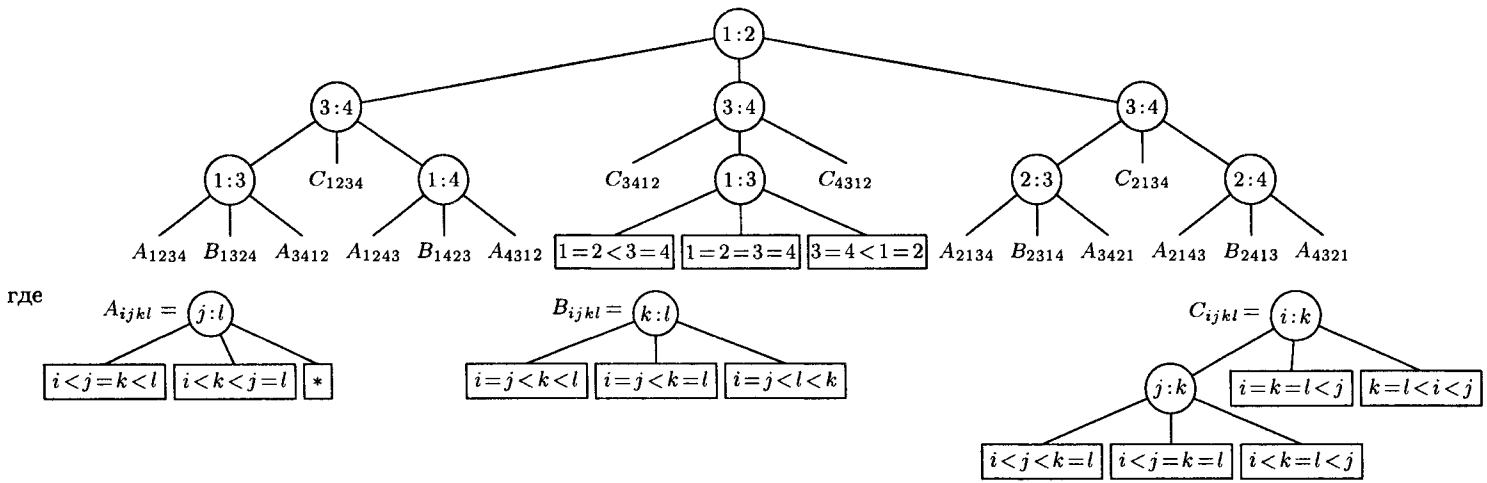


Рис. А–2. Решение упр. 8.

15. Если $\lceil \lg \frac{3}{4}n \rceil = \lg(\frac{3}{4}n) + \theta$, то $F(n) = n\lg n - (3 - \lg 3)n + n(\theta + 1 - 2^\theta) + O(\log n)$. Если $\lceil \lg n \rceil = \lg n + \theta$, то $B(n) = n\lg n - n + n(\theta + 1 - 2^\theta) + O(\log n)$. [Обратите внимание на то, что $\lg n! = n\lg n - n/(\ln 2) + O(\log n)$; $1/(\ln 2) \approx 1.443$; $3 - \lg 3 \approx 1.415$.]

17. Число случаев, когда $b_k < a_p < b_{k+1}$, равно

$$\binom{m-p+n-k}{m-p}\binom{p-1+k}{p-1},$$

а число случаев, когда $a_j < b_q < a_{j+1}$, равно

$$\binom{n-q+m-j}{n-q}\binom{q-1+j}{q-1}.$$

18. Нет, поскольку мы рассматриваем всегда наименее эффективную ветвь под каждым узлом-сравнением. Одна из более эффективных ветвей могла бы оказаться более трудоемкой.

20. Пусть L — максимальный номер уровня, на котором есть внешние узлы, а l — такой минимальный номер. Если $L \geq l + 2$, то на уровне L можно удалить два узла и поместить их под некоторым узлом на уровне l; в результате длина внешнего пути сократится на $l + 2L - (L - 1 + 2(l + 1)) = L - l - 1 \geq 1$. Обратно, если $L \leq l + 1$, то пусть k внешних узлов находятся на уровне l и $N - k$ узлов — на уровне $l + 1$, где $0 < k \leq N$. Как показано в упр. 2.3.4.5–3, $k2^{-l} + (N - k)2^{-l-1} = 1$; следовательно, $N + k = 2^{l+1}$. Из неравенств $2^l \leq N < 2^{l+1}$ вытекает, что $l = \lfloor \lg N \rfloor$; отсюда определяется значение k, а также получается длина внешнего пути (34).

21. Пусть $r(x)$ — корень правого поддерева узла x. Все поддеревья будут иметь минимальную высоту тогда и только тогда, когда $\lceil \lg t(l(x)) \rceil \leq \lceil \lg t(x) \rceil - 1$ и $\lceil \lg t(r(x)) \rceil \leq \lceil \lg t(x) \rceil - 1$ для всех x. Первое из них эквивалентно условию $2t(l(x)) - t(x) \leq 2^{\lceil \lg t(x) \rceil} - t(x)$, а второе — условию $t(x) - 2t(l(x)) \leq 2^{\lceil \lg t(x) \rceil} - t(x)$.

22. Согласно упр. 20 четыре условия, $\lfloor \lg t(l(x)) \rfloor$, $\lfloor \lg t(r(x)) \rfloor \geq \lfloor \lg t(x) \rfloor - 1$ и $\lceil \lg t(l(x)) \rceil$, $\lceil \lg t(r(x)) \rceil \leq \lceil \lg t(x) \rceil - 1$, необходимы и достаточны. Рассуждая, как в упр. 21, можно доказать, что они эквивалентны сформулированным в условии этого упражнения неравенствам. [См. Martin Sandelius, *AMM* **68** (1961), 133–134.] Распространение этого решения на общий случай приводится в упр. 33.

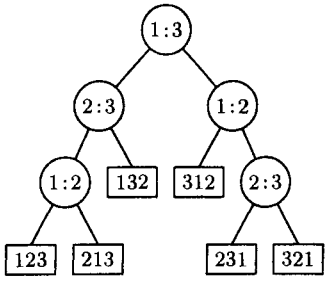
23. При сортировке посредством вставок в несколько списков предполагается, что ключи равномерно распределены в известном диапазоне, значит, он не принадлежит к числу методов "чистых сравнений", удовлетворяющих ограничениям, которые рассматриваются в этом разделе.

24. Действуйте сначала, как при сортировке пяти элементов, до тех пор, пока не получите одну из конфигураций (6). В первых трех случаях завершите сортировку пяти элементов при помощи еще двух сравнений, после чего вставьте шестой элемент f. В последнем случае нужно сначала сравнить $f:b$, вставить f в главную цепочку, а затем вставить c. [Picard, *Théorie des Questionnaires*, page 116.]

25. Поскольку $N = 7! = 5040$ и $q = 13$, было бы $8192 - 5040 = 3152$ внешних узла на уровне 12 и $5040 - 3152 = 1888$ внешних узлов на уровне 13.

26. В работе Ľ. Kollár, *Lecture Notes in Comp. Sci.* **233** (1986), 449–457, представлен прекрасный способ проверки утверждения, что оптимальный метод сортировки дает путь длиной 62 416.

27. Это единственный способ распознать две наиболее часто встречающиеся перестановки за два сравнения, несмотря на то что первому сравнению соответствует разбиение с вероятностью .27/.73.

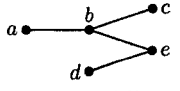


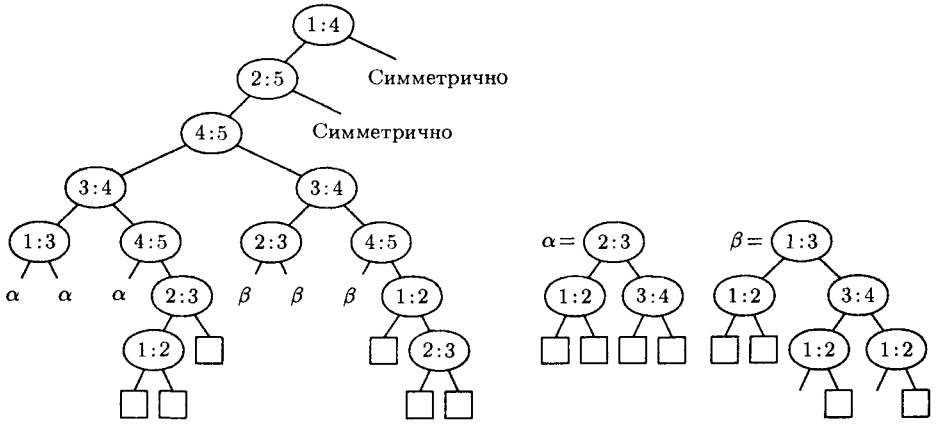
28. Лунь Куань (Lun Kwan) составил программу длиной 873 строки, среднее время работы которой равно $38.925u$. Максимальное время ее работы равно $43u$; по-видимому, оно оптимально, поскольку как раз столько времени требуется для выполнения 7 сравнений, 7 проверок, 6 загрузок и 5 записей в память.

29. Не выполнив $S(n)$ сравнений, невозможно дать однозначный ответ, какой является перестановка: четной или нечетной. В самом деле, если предположить, что в результате выполнения некоторого числа сравнений мы "сузили" задачу до такой степени, что осталось всего два возможных исхода в зависимости от того, будет ли a_i меньше или больше a_j, где i и j — некоторые индексы, то один из этих возможных исходов — четная перестановка, другой — нечетная. [С другой стороны, для решения этой задачи *существует* алгоритм с временем работы $O(n)$; он просто подсчитывает количество циклов и вовсе не содержит сравнений; см. упр. 5.2.2–2.]

30. Возьмите оптимальное дерево сравнений высотой $S(n)$. Двигаясь сверху вниз, меняйте местами $i \leftrightarrow j$ в правом поддереве узла $i{:}j$. В полученном дереве, которое интерпретируется как дерево сравнений-обменов, каждый концевой узел определяет единственную перестановку, а ее можно рассортировать не более чем за $n-1$ дополнительных сравнений-обменов (это показано в упр. 5.2.2–2). [Идея дерева сравнений-обменов принадлежит Т. Н. Хиббарду (T. N. Hibbard).]

31. Необходимо не менее 8 сравнений-обменов, поскольку в любом дереве высотой 7 существует ветвь, которая после 4 шагов приведет к конфигурации, где $a \neq 1$ (или двойственной по отношению к ней). Эту конфигурацию нельзя рассортировать за три операции сравнения-обмена. С другой стороны, ниже показано дерево, на котором достигается искомая нижняя граница оценки (а также, быть может, минимальное *среднее* число сравнений-обменов).





33. К любому дереву порядка x с разрешением 1 можно применить простую операцию для получения другого дерева, длина взвешенного пути которого не превышает длины пути исходного дерева, причем: (a) существует такое число k, что все внешние узлы лежат на уровнях k и $k-1$, (b) не более чем один внешний узел содержит нецелое значение, и если таковой имеется, то он находится на уровне k. Длина взвешенного пути любого такого дерева имеет указанное значение, так что оно должно быть минимальным. Обратно, если в вещественнозначном дереве поиска выполняются условия (iv) и (v), то, применив индукцию, можно показать, что длина взвешенного пути имеет указанное значение, поскольку для этого параметра существует простая формула, выражающая ее через длины путей двух поддеревьев корня.

35. Ключ к решению этой задачи содержится в безуспешных экспериментах, описанных в работе Knuth, Kaehler, *Information Processing Letters* **1** (1972), 173–176.

36. См. *Математические заметки* **4** (1968), 511–518. Обзор достижений в решении этой задачи представлен в работе S. Felsner, W. T. Trotter, *Combinatorics, Paul Erdős is Eighty* **1** (1993), 145–157. Там же доказано, что мы всегда можем получить $1 \le T(G_1)/T(G_2) \le \rho$, где константа ρ чуть-чуть меньше 8/3.

РАЗДЕЛ 5.3.2

1. $S(m+n) \le S(m) + S(n) + M(m,n)$.

2. Внутренний узел, который является k-м по счету при отношении порядка, симметричном к исходному, соответствует сравнению $A_1 : B_k$.

3. Стратегия $\mathrm{B}(1,l)$ не лучше стратегии $\mathrm{A}(1,l{+}1)$, а стратегия $\mathrm{B}'(1,l)$ не превосходит стратегии $\mathrm{A}'(1,l{-}1)$. Следовательно, нужно разрешить рекуррентное соотношение

$$.M.(1,n) = \min_{1\le j\le n} \max\bigl(\max_{1\le l\le j}(1{+}.M.(1,l{-}1)),\ \max_{j\le l\le n}(1{+}.M.(1,n{-}l))\bigr), \quad n \ge 1;$$

$$.M.(1,0) = 0.$$

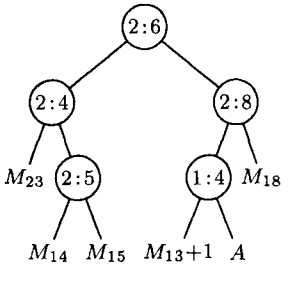
Нетрудно проверить, что этому соотношению удовлетворяет решение $\lceil \lg(n+1) \rceil$.

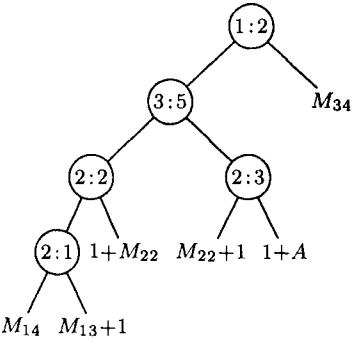
4. Нет [см. C. Christen, *FOCS* **19** (1978), 259–266].

6. При $j = i+1$ можно воспользоваться стратегией $\mathrm{A}'(i,i{+}1)$, за исключением случая $i \le 2$. При $j \ge i+2$ можно применить стратегию $\mathrm{A}(i,i{+}2)$.

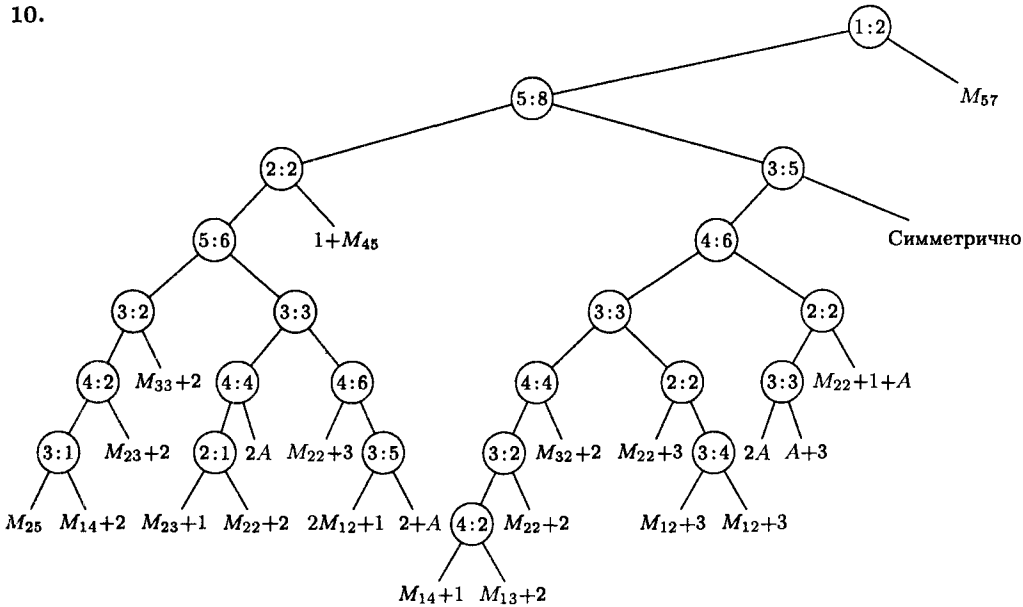
7. Чтобы вставить $k+m$ элементов в последовательность n других элементов, вставьте независимо k элементов и m элементов. (Если k и m велики, то можно применить более совершенную процедуру; см. упр. 19.)

8. На следующих диаграммах $i{:}j$ означает сравнение $A_i : B_j$; через M_{ij} обозначено слияние i элементов с j элементами за $M(i,j)$ шагов, а через A — сортировка конфигурации или за три шага.





10.



11. Воспользуемся указанием: положим $n = g_t$. Можем считать, что $t \geq 6$. Без потери общности можно первым сравнением сделать $A_2 : B_j$. Если $j > g_{t-1}$, то исход $A_2 < B_j$ потребует еще $\geq t$ шагов. Если $j \leq g_{t-1}$, то исход $A_2 < B_j$ не вызовет трудностей, и поэтому необходимо исследовать лишь случай $A_2 > B_j$. Больше всего информации мы получаем при $j = g_{t-1}$. Если $t = 2k+1$, то можно было бы слить A_2 с $g_t - g_{t-1} = 2^{k-1}$ элементами, превышающими $B_{g_{t-1}}$, и слить A_1 с остальными g_{t-1} элементами, но для этого понадобилось бы еще $k + (k+1) = t$ шагов. С другой стороны, если $n = g_t - 1$, то можно было бы слить A_2 с $2^{k-1} - 1$ элементами, а затем A_1 — с n элементами еще за $(k-1) + (k+1)$ шагов. Следовательно, $M(2, g_t - 1) \leq t$.

Случай $t = 2k$ значительно сложнее; обратите внимание на то, что $g_t - g_{t-1} \geq 2^{k-2}$. Предположим, что если $A_2 > B_{g_{t-1}}$, то мы будем сравнивать $A_1 : B_j$. Если $j > 2^{k-1}$, то исход $A_1 < B_j$ потребует еще $k + (k-1)$ сравнений (слишком много). Если $j \leq 2^{k-1}$, то можно, как и раньше, доказать, что выбор $j = 2^{k-1}$ даст больше всего информации. В случае $A_1 > B_{2^{k-1}}$ можно было бы сравнить также A_1 с $B_{2^{k-1}+2^{k-2}}$, затем с $B_{2^{k-1}+2^{k-2}+2^{k-3}}$; поскольку $2^{k-1}+2^{k-2}+2^{k-3} > g_{t-1}$, остается лишь слить $\{A_1, A_2\}$ с $n-(2^{k-1}+2^{k-2}+2^{k-3})$ элементами. Разумеется, вовсе не обязательно сразу же выполнять какие-либо сравнения с A_1; можно вместо этого сравнить $A_2 : B_{n+1-j}$. Если $j \leq 2^{k-3}$, то рассматриваем случай $A_2 < B_{n+1-j}$, если же $j > 2^{k-3}$, то рассматриваем $A_2 > B_{n+1-j}$. Этот последний случай требует еще не менее $(k-2) + (k+1)$ шагов. Продолжая рассуждение, обнаруживаем, что *единственный* потенциально плодотворный путь — это $A_2 > B_{g_{t-1}}$, $A_2 < B_{n+1-2^{k-3}}$, $A_1 > B_{2^{k-1}}$, $A_1 > B_{2^{k-1}+2^{k-2}}$, $A_1 > B_{2^{k-1}+2^{k-2}+2^{k-3}}$, но тогда остается еще ровно g_{t-5} элементов! Обратно, если $n = g_t - 1$, то этот путь приводит к желаемому результату. [*Acta Informatica* 1 (1971), 145–158.]

12. Первое сравнение должно быть либо $\alpha : X_k$, где $1 \leq k \leq i$, либо (симметрично) $\beta : X_{n-k}$, где $1 \leq k \leq j$. В первом случае при исходе $\alpha < X_k$ остается выполнить еще $R_n(k-1, j)$ сравнений; исход $\alpha > X_k$ приводит к задаче сортировки $\alpha < \beta$, $Y_1 < \cdots < Y_{n-k}$, $\alpha < Y_{i-k+1}$, $\beta > Y_{n-k-j}$, где $Y_r = X_{r-k}$.

13. См. *Computers in Number Theory* (New York: Academic Press, 1971), 263–269.

14. *SICOMP* **9** (1980), 298–320. Полное решение для случая $M(4,n)$ получено спустя непродолжительное время в работе J. Schulte, Mönting, *Theor. Comp. Sci.* **14** (1981), 19–37, где также предлагается решение для случая $M(5,n)$.

15. Удваивать m до тех пор, пока результат не превзойдет n. Для этого придется выполнить $\lfloor \lg(n/m) \rfloor + 1$ операций удваивания.

16. При всех (m,n), за исключением $(m,n) = (2,8), (3,6), (3,8), (3,10), (4,8), (4,10), (5,9), (5,10)$. При этих значениях число сравнений на единицу больше оптимального.

17. Предположим, $m \le n$, и пусть $t = \lg(n/m) - \theta$. Тогда $\lg \binom{m+n}{m} > \lg n^m - \lg m! \ge m \lg n - (m \lg m - m + 1) = m(t+\theta) + m - 1 = H(m,n) + \theta m - \lfloor 2^\theta m \rfloor \ge H(m,n) + \theta m - 2^\theta m \ge H(m,n) - m$. (Неравенство $m! \le m^m 2^{1-m}$ является следствием того, что $k(m-k) \le (m/2)^2$ при $1 \le k < m$.)

19. Слейте сначала $\{A_1, \ldots, A_m\}$ с $\{B_2, B_4, \ldots, B_{2\lfloor n/2 \rfloor}\}$. Тогда останется вставить нечетные элементы B_{2i-1} в последовательность a_i элементов A, $1 \le i \le \lceil n/2 \rceil$, где $a_1 + a_2 + \cdots + a_{\lceil n/2 \rceil} \le m$. При выполнении этого последнего действия на каждое значение i придется выполнить a_i операций, так что для завершения сортировки достаточно выполнить еще не более m сравнений.

20. Применить неравенство (12).

22. В работе R. Michael Tanner, *SICOMP* **7** (1978), 18–38, показано, что в алгоритме "фрактальных вставок" используется в среднем не более $1.06 \lg \binom{m+n}{m}$ сравнений. В работе Ľ. Kollár, *Computers и Artificial Int.* **5** (1986), 335–344, проанализировано поведение "в среднем" алгоритма H.

23. Соперник имеет матрицу X размера $n \times n$, элементы которой x_{ij} вначале все равны единице. Когда алгоритм спрашивает, имеется ли равенство $A_i = B_j$, соперник устанавливает x_{ij} равным нулю. Он отвечает "нет" до тех пор, пока перманент матрицы X не станет равным нулю. В этом случае соперник отвечает "да" (ему ничего не остается делать, иначе выполнение алгоритма немедленно завершится!) и исключает из матрицы X строку i и столбец j; полученная матрица $(n-1) \times (n-1)$ будет иметь ненулевой перманент. Соперник продолжает и дальше действовать таким образом, пока у него не останется матрица 0×0.

Если перманент близок к нулю, можно перекомпоновать строки и столбцы таким образом, что $i = j = 1$ и все единицы будут на главной диагонали матрицы. Таким образом перманент обращается в нуль, если $x_{11} \leftarrow 1$; значит, мы должны получить $x_{1k}x_{k1} = 0$ для всех $k > 1$. Отсюда следует, что, по крайней мере, n нулей удаляются при первом ответе "да" соперника, а $n-1$ — при втором ответе и т. д. Выполнение алгоритма завершится лишь после получения n положительных ответов на неизбыточные вопросы и после того как мы зададим, по меньшей мере, $n + (n-1) + \cdots + 1$ вопросов [*JACM* **19** (1972), 649–659]. Аналогичные рассуждения приведут нас к заключению, что необходимо $n + (n-1) + \cdots + (n-m+1)$ вопросов для того, чтобы выяснить, что $A \subseteq B$ при $|A| = m \le n = |B|$.

24. Грубое предварительное слияние потребует не более $m + q - 1$ слияний, а каждая последующая процедура вставки — не более t. Эта верхняя оценка не может быть уменьшена. Таким образом, максимальное число сравнений будет тем же, что и в алгоритме H (см. (19)).

25. В общем, сложность этой задачи такова же, как и для отдельного случая, когда каждый элемент x_{ij} — это либо 0, либо 1 и $x = \frac{1}{2}$. Тогда каждое сравнение эквивалентно анализу бита x_{ij} и мы стремимся определить всю матрицу, просматривая только младшие разряды. Любая задача сортировки (1) соответствует матрице 0–1, если установить $x_{ij} = [A_i > B_{n+1-j}]$. (В работе N. Linial, M. Saks, *J. Algorithms* **6** (1985), 86–103, этот вывод приписывается Дж. Ширеру (J. Shearer). Аналогичный результат связывает сортировку и поиск при любом частичном упорядочении.)

РАЗДЕЛ 5.3.3

1. Игрок *11* проиграл игроку *05*, поэтому известно, что игрок *13* слабее, чем игроки *05*, *11* и *12*.

2. Пусть x есть t-й в порядке убывания элемент и пусть S есть множество всех элементов y, таких, что выполненных сравнений недостаточно для доказательства того, что $x < y$ и $y < x$. Существуют перестановки, удовлетворяющие всем выполненным сравнениям, но в них все элементы S меньше, чем x; действительно, мы можем потребовать, чтобы все элементы S были меньше, чем x, и вложить полученное частичное упорядочение в линейное упорядочение. Аналогично существуют допустимые перестановки, в которых все элементы S больше, чем x. Следовательно, мы не можем знать положение x, если S не пусто.

3. Соперник может считать проигравшего в первом сравнении слабейшим среди всех игроков.

4. Предположим, что наибольшие $(t-1)$ элементов — это $\{a_1, \ldots, a_{t-1}\}$. Любой путь определения наибольших t-элементов на дереве сравнений, совместимый с этим предположением, должен включать, по меньшей мере, $n-t$ сравнений для определения наибольшего из оставшихся $n-t+1$ элементов. Каждый из таких путей должен иметь не менее $n-t$ точек выбора альтернатив, т. е. их в сумме существует не менее 2^{n-t}. Таким образом, каждый из $n^{\underline{t-1}}$ выборов наибольших $(t-1)$ элементов должен появиться не менее чем в 2^{n-t} листьях дерева.

5. Действительно, $W_t(n) \le V_t(n) + S(t-1)$, как следует из упр. 2.

6. Пусть $g(l_1, l_2, \ldots, l_m) = m-2+\lceil \lg(2^{l_1}+2^{l_2}+\cdots+2^{l_m})\rceil$, и допустим, что $f = g$ для $l_1+l_2+\cdots+l_m+2m < N$. Докажем, что $f = g$, если $l_1+l_2+\cdots+l_m+2m = N$. Мы можем считать, что $l_1 \ge l_2 \ge \cdots \ge l_m$. Существует лишь несколько способов выполнения первого сравнения.

Стратегия $\mathrm{A}(j,k)$ при $j < k$. Сравнить наибольший элемент группы j с наибольшим элементом группы k. Это дает соотношение

$$\begin{aligned} f(l_1, \ldots, l_m) &\le 1 + g(l_1, \ldots, l_{j-1}, l_j+1, l_{j+1}, \ldots, l_{k-1}, l_{k+1}, \ldots, l_m) \\ &= g(l_1, \ldots, l_{j-1}, l_j, l_{j+1}, \ldots, l_{k-1}, l_j, l_{k+1}, \ldots, l_m) \ge g(l_1, \ldots, l_m). \end{aligned}$$

Стратегия $\mathrm{B}(j,k)$ при $l_k > 0$. Сравнить наибольший элемент группы j с одним из меньших элементов группы k. Это дает соотношение $f(l_1, \ldots, l_m) \le 1 + \max(\alpha, \beta) = 1 + \beta$, где $\alpha = g(l_1, \ldots, l_{j-1}, l_{j+1}, \ldots, l_m) \le g(l_1, \ldots, l_m) - 1$, $\beta = g(l_1, \ldots, l_{k-1}, l_k-1, l_{k+1}, \ldots, l_m) \ge g(l_1, \ldots, l_m) - 1$. *Стратегия* $\mathrm{C}(j,k)$ при $j \le k$, $l_j > 0$, $l_k > 0$. Сравнить один из меньших элементов группы j с одним из меньших элементов группы k. Соответствующим соотношением будет $f(l_1, \ldots, l_m) \le 1 + g(l_1, \ldots, l_{k-1}, l_k - 1, l_{k+1}, \ldots, l_m) \ge g(l_1, \ldots, l_m)$.

Значение $f(l_1, \ldots, l_m)$ найдем, взяв минимум правой части среди всех этих стратегий; следовательно, $f(l_1, \ldots, l_m) \ge g(l_1, \ldots, l_m)$. Если $m > 1$, то стратегия $\mathrm{A}(m-1, m)$ показывает, что $f(l_1, \ldots, l_m) \le g(l_1, \ldots, l_m)$, поскольку $g(l_1, \ldots, l_{m-1}, l_m) = g(l_1, \ldots, l_{m-1}, l_{m-1})$, если $l_1 \ge \cdots \ge l_m$. (*Доказательство.* $\lceil \lg(M + 2^a)\rceil = \lceil \lg(M + 2^b)\rceil$ при $0 \le a \le b$, если M есть положительное кратное 2^b.) Если $m = 1$, используйте стратегию $\mathrm{C}(1,1)$.

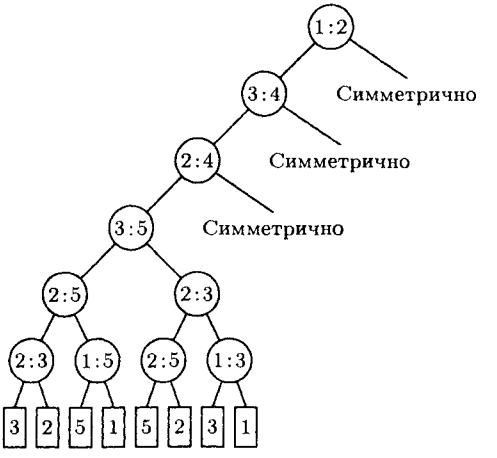
[В статье С. С. Кислицына определена оптимальная стратегия $\mathrm{A}(m-1, m)$ и вычислено $f(l, l, \ldots, l)$ в замкнутом виде; общая формула для f и это упрощенное доказательство были получены Флойдом в 1970 году.]

7. При $j > 1$, если $j+1$ находится в α', c_j равно 1 плюс число сравнений, необходимых для выбора следующего в порядке убывания элемента α'. Аналогично в случае, если $j+1$ находится в α'', c_1 всегда равно 0, так как деревья в конце всегда выглядят одинаково.

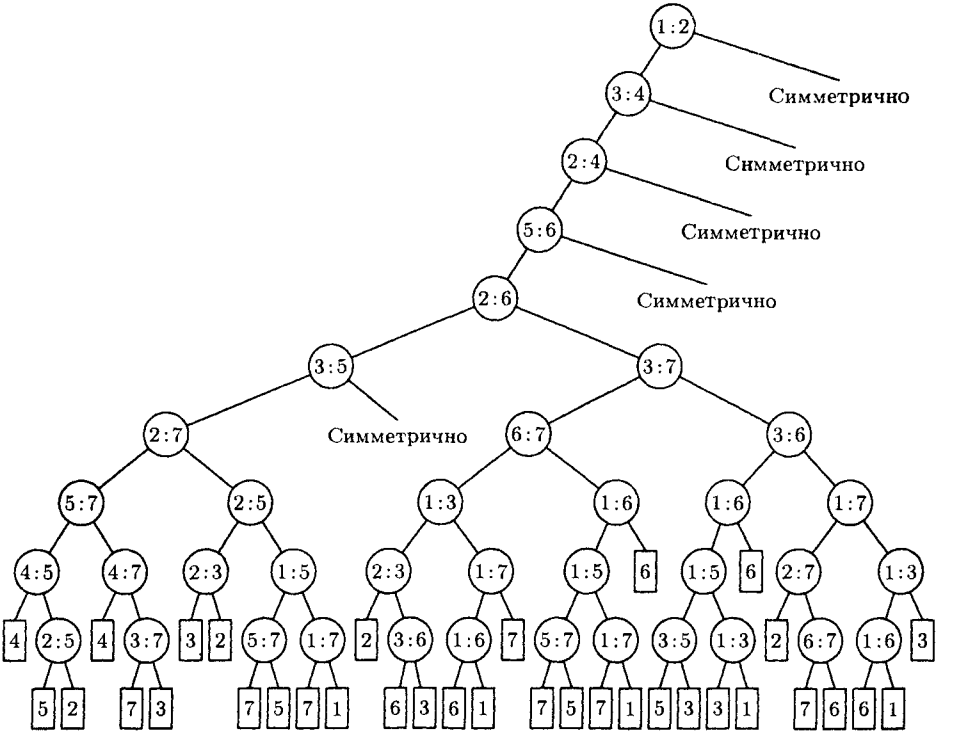
8. Другими словами, существует ли расширенное бинарное дерево с n внешними узлами, такое, что сумма расстояний от корня до $t-1$ наиболее удаленных внутренних узлов меньше, чем соответствующая сумма для полного бинарного дерева? Ответ на этот вопрос

отрицательный, поскольку (как нетрудно показать) k-й в порядке убывания элемент $\mu(\alpha)$ больше или равен $\lfloor \lg(n-k) \rfloor$ при всех α.

9. (Для всех путей используются шесть сравнений, однако эта процедура не оптимальна для $\bar{V}_3(5)$.)



10. (Медиана найдена вручную путем проб и ошибок; использовано упр. 6, чтобы упростить нахождение полезных линий.)



11. См. *Information Processing Letters* **3** (1974), 8–12.

12. После удаления наименьшего из $\{X_1, X_2, X_3, X_4\}$ получаем конфигурацию $\bullet\!\!-\!\!\bullet$ плюс $n-3$ изолированных элементов; третий из них может быть найден еще за $V_3(n-1)-1$ шагов.

13. Найдя медиану первых $f(n)$ элементов, скажем X_j, сравним ее с каждым из остальных; это даст разбиение всех элементов на приблизительно $n/2-k$ меньших, чем X_j, и $n/2+k$ больших, чем X_j, при некотором k. Остается найти $|k|$-й в порядке убывания или возрастания элемент большего множества, что требует еще $n/2+O(|k|\log n)$ сравнений. Среднее значение $|k|$ равно $O(1/\sqrt{n})+O(n/\sqrt{f(n)})$ (рассмотрите точки, равномерно распределенные на интервале $[0\,..\,1]$). Обозначим через $T(n)$ среднее число сравнений для $f(n)=n^{2/3}$; имеем $T(n)-n=T(n^{2/3})-n^{2/3}+n/2+O(n^{2/3})$; отсюда следует искомый результат.

Интересно, что при $n=5$ этот метод требует только $5\frac{13}{15}$ сравнений в среднем, что несколько лучше, чем дерево из упр. 9.

14. В общем случае t наибольших элементов можно найти за $U_t(n)\le V_t(n-1)+1$ сравнений, если определить t-й в порядке убывания элемент из $\{X_1,\ldots,X_{n-1}\}$ и сравнить его с X_n (имея в виду упр. 2). (Киркпатрик доказал, что (12) является нижней оценкой для $U_t(n+1)-1$. При бóльших t найдена более точная оценка $U_t(n)$ [см. J. W. John, *SICOMP* **17** (1988), 640–647].)

15. Требуется $\min(t, n+1-t)$ слов памяти. Предположим, что $t\le n+1-t$. Если мы не сохраним все первые t слов, когда они читались в первый раз, то можем потерять t-й в порядке убывания элемент, зависящий от последующих значений, все еще не известных нам. Обратно, t ячеек достаточно, так как мы можем сравнивать вновь введенный элемент с предыдущим t-м, запоминая значение в регистре тогда и только тогда, когда он больше.

16. Перед началом выполнения алгоритмы $(a,b,c,d)=(n,0,0,0)$, а после завершения получается четверка $(0,1,1,n-2)$. Если соперник избегает непредвиденных исходов, то после каждого сравнения (a,b,c,d) может перейти либо в себя, либо в

$$\begin{aligned}
(a-2,\ b+1,\ c+1,\ d), &\qquad \text{если } a\ge 2;\\
(a-1,\ b,\ c+1,\ d) \text{ или } (a-1,\ b+1,\ c,\ d), &\qquad \text{если } a\ge 1;\\
(a,\ b-1,\ c,\ d+1), &\qquad \text{если } b\ge 2;\\
(a,\ b,\ c-1,\ d+1), &\qquad \text{если } c\ge 2.
\end{aligned}$$

Отсюда следует, что требуется $\lceil\frac{3}{2}a\rceil+b+c-2$ сравнений, чтобы из (a,b,c,d) получить $(0,1,1,a+b+c+d-2)$. [*См. CACM* **15** (1972), 462–464. В работе *FOCS* **16** (1975), 71–74, Пол доказал, что этот алгоритм также минимизирует *среднее* число сравнений.]

17. Используйте (6) сначала для наибольшего элемента, затем — для наименьшего. Обратите внимание на то, что $\lfloor n/2\rfloor$ сравнений общие для обоих случаев.

18. $V_t(n)\le 18n-151$ при всех достаточно больших n.

21. **Шаг 0**. Постройте два дерева турниров с выбыванием размерами 2^k и 2^{k-t+1}.

Шаг j для $1\le j\le t$. (К этому моменту уже выведено $j-1$ наибольших элементов. Оставшиеся элементы вместе с набором фиктивных, каждый из которых равен $-\infty$, появятся теперь в двух деревьях, A и B, где A имеет 2^k листьев, а B — 2^{k-t+j}.) Пусть a — победитель в A, и предположим, что a выиграл у a_0, a_1, ..., a_{k-1}, где a_l — победитель среди 2^l элементов. Аналогично пусть b и b_0, b_1, ..., $b_{k-t+j-1}$ — победитель и обладатель второго места в B. Если $j=t$, вывести $\max(a,b)$ и прекратить выполнение. В противном случае "нарастить" другой уровень в нижней части B, вставив 2^{k-t+j} фиктивных элементов, каждый из которых считается проигравшим в первой встрече с участником B. (Наша стратегия — влить B в A, если возможно, заменив им поддерево

A' в A, которое содержит a_0, a_1, ..., a_{k-t+j}. Обратите внимание на то, что A', как и только что подросшее B, есть дерево турнира с выбыванием, в котором $2^{k-t+j+1}$ листьев.) Сравните b с $a_{k-t+j+1}$, затем сравните победителя с $a_{k-t+j+2}$ и так до тех пор, пока не будет найдено $c = \max(b, a_{k-t+j-1}, \ldots, a_{k-1})$. Случай 1, $b < c$: вывести a и поменять B с A'. Случай 2, $b = c$ и $b < a$: вывести a и поменять B с A'. Случай 3, $b = c$ и $b > a$: вывести b. После отработки этих трех случаев мы останемся с деревьями A и B (возможно, новыми), в которых победитель B только что ушел на выход. Удалите этот элемент из B и замените его элементом $-\infty$, выполнив все сравнения, необходимые для восстановления структуры дерева (как в методе выбора из дерева). Этим завершится шаг j.

На шаге 0 выполняется $2^k - 1 + 2^{k+1-t} - 1$ сравнений, а на шаге t выполняется одно. На каждом из шагов 1, 2, ..., $t-1$ выполняется не более $k-1$ сравнений, кроме случая 2, когда их может быть k. Но если только возникает случай 2, мы сохраним одно сравнение на следующий раз для случая 1 или случая 2, поскольку a_0 тогда окажется равным $-\infty$. Таким образом, суммарно на шагах от 1 до $t-1$ выполняется не более $(t-1)(k-1)+1$ сравнений.

Как следует из результатов упр. 3, имеем $W_t(n) \le n + (t-1)(k-1)$ для всех $n \le 2^k + 2^{k+1-t}$, если $k \ge t \ge 2$. Если $n \ge 2^k + t - 2$, результаты упр. 4 говорят, что $W_t(n) \ge n - t + \lceil \lg(2^k + t - 2) \frac{t-1}{} \rceil$, а это равно $n - t + (t-1)k + 1$, если $t \ge 3$. Таким образом, данный метод оптимален при $2^k + t - 2 \le n \le 2^k + 2^{k+1-t}$, если $k \ge t \ge 3$ (а также для некоторых меньших значений n, если t больше).

Аналогичный метод, в котором используются резервные элементы вместо $-\infty$ при перестройке B в конце шагов 1, ..., $t-2$, доказывает, что $V_t(n) \le n + (t-1)(k-1)$, когда $n \le 2^k + 2^{k+1-t} + t - 2$ и $k \ge t \ge 3$ (см. доказательство формулы (11)). [См. *J. Algorithms* **5** (1984), 557–578.]

22. В общем случае, если $2^r \cdot 2^k < n + 2 - t \le (2^r + 1) \cdot 2^k$ и $t < 2^r \le 2t$, такая процедура с $t+1$ деревьями с выбыванием размером 2^k использует на $\lfloor (t-1)/2 \rfloor$ сравнений меньше, чем (11), так как, по крайней мере, столько сравнений, примененных в (ii) для нахождения минимума, могут быть вновь использованы в (iii).

23. Как следует из (15), количество $V_{\lceil n/2 \rceil}(n)/n$ ограничено снизу значением 2 при $n \to \infty$; но в работе D. Dor, Ph. D. thesis, Tel Aviv University, 1995, показано, что в действительности нижняя граница значительно выше 2. Дор и Ю. Звик (U. Zwick) также нашли верхнюю границу, которая оказалась равной 2.942 [*SODA* **6** (1995), 28–37]; они доказали, что асимптота для верхней оценки есть

$$V_{\alpha n}(n) \le \left(1 + \alpha \lg \frac{1}{\alpha} + O\left(\alpha \log\log \frac{1}{\alpha}\right)\right) n,$$

что не очень отличается от (15), если α достаточно мало [*Combinatorica* **16** (1996), 41–58].

24. Поскольку $W_t(n) = n + O(t \log n)$, как следует из выражения (6), утверждение, которое содержится в указании, определенно, справедливо, если $t \le \sqrt{n/\ln n}$. Предположим, что это утверждение выполняется для n, и пусть u и v имеют ранги $t_- = \lfloor t - \sqrt{t \ln n} \rfloor$ и $t_+ = \lceil t + \sqrt{t \ln n} \rceil$ среди первых n из расположенных в случайном порядке $2n$ элементов. (Наименьший элемент имеет ранг 1.) Сравните другие n элементов с v и сравните те из них, которые меньше v, также с u. Вероятность p_s того, что элемент x ранга t в первых n имеет ранг s среди всех, равна $\binom{s-1}{t-1}\binom{2n-s}{n-t}/\binom{2n}{n}$. Среднее значение s есть $\sum s p_s = \frac{2n+1}{n+1} t$; это среднее число элементов $< x$; следовательно, среднее число сравнений с u равно $\binom{n}{n+1} t_+ = t + O(n \log n)^{1/2}$. Пусть u и v имеют ранги s_- и s_+ среди всех $2n$ элементов и пусть $T_- = \lfloor 2t - \sqrt{2t \ln 2n} \rfloor$, $T_+ = \lceil 2t + \sqrt{2t \ln 2n} \rceil$. Если $s_- \le T_-$ и $s_+ \ge T_+$, можно отыскать элементы рангов T_- и T_+ посредством выбора $s_+ - s_- + 1$ элементов между u и v. Докажем, что очень маловероятно, чтобы оказалось $s_- > T_-$ или $s_- < T_- - 2\sqrt{n \ln n}$, или $s_+ < T_+$, или $s_+ >$

$T_+ + 2\sqrt{n \ln n}$; таким образом, почти всегда будет достаточно $O(n \log n)^{1/2}$ дополнительных сравнений. Из указания следует индукция по n, если мы сможем показать, что "очень маловероятно" означает "с вероятностью $O(n^{-1-\epsilon})$ для всех достаточно больших n".

Обратите внимание, что $p_{s+1}/p_s = s(n-s+t)/(s+1-t)(2n-s)$ уменьшается по мере того, как s увеличивается от t до $n+t$ и это отношение ≤ 1 тогда и только тогда, когда $s \ge 2n(t-1)/(n-1)$; оно $\le 1 - \frac{1}{2}cn^{-1/2} + O(n^{-1})$, когда $s = \overline{s}(c) = 2t + ct(n-t)/n^{3/2}$. Таким образом, вероятность того, что $s \ge \overline{s}(c)$, равна $\le 2c^{-1}n^{1/2}p_{\overline{s}(c)}(1 + O(n^{-1/2}))$. Аналогично $p_{s-1}/p_s < 1 - \frac{1}{2}cn^{-1/2} - O(n^{-1})$, если $s = \underline{s}(c) = 2t - 1 - c(t-1)(n+1-t)/n^{3/2}$, так что $s \le \underline{s}(c)$ с вероятностью $\le 2c^{-1}n^{1/2}p_{\underline{s}(c)}(1 + O(n^{-1/2}))$. В тех случаях, которые нас интересуют, подходящие значения c есть $\ge .55n^{3/2}(\ln n)^{1/2}t^{-1/2}(n-t)^{-1}$ для всех больших n, а аппроксимация Стирлинга дает, что $p_{\overline{s}(c)}$ и $p_{\underline{s}(c)}$ равны

$$O(n^{1/2}s^{-1/2}(2n-s)^{-1/2})\exp(-2sc^2(n-t)^2/n^3 - 2(2n-s)c^2t^2/n^3)$$
$$\le O(t^{-1/2}\exp(-4t(n-t)c^2/n^2)) \le O(t^{-1/2}n^{-1.2}).$$

Это подтверждает, что вероятность $O(n^{-1.2}(\log n)^{1/2})$ действительно очень мала. [Аналогичная схема имеется и в *CACM* **18** (1975), 165–172, но представленный в этой работе анализ некорректен.]

25. Имея заданный алгоритм выбора и перестановку π множества $\{1, \ldots, n\}$, организуем для каждой i-й перестановки отдельный счет (аккумулятор). После каждого сравнения $\pi_i : \pi_j$ добавим 1 на счет π_i, если $|\pi_i - t| > |\pi_j - t|$; если же $|\pi_i - t| = |\pi_j - t|$, отнесем по $\frac{1}{2}$ на счет обеих. Отнесение на счет π_i называется *полезным*, если $\pi_i < \pi_j \le t$ или $\pi_i > \pi_j \ge t$; в противном случае оно *бесполезно*. Пусть x_k — итоговый счет для k. Тогда общее число сравнений равно $x_1 + \cdots + x_n$. Очевидно, что $x_t = 0$; но $x_k \ge 1$ для всех $k \ne t$, поскольку каждый элемент, отличный от t, имеет полезный счет. Докажем, что $\mathrm{E}\, x_{t+k} + \mathrm{E}\, x_{t-k} \ge 3$ при $0 < k < t$.

Пусть $A_k(\pi) =$ [первое добавление на счет $t+k$ было бесполезным]. Тогда $A_k(\pi) = 1 - A_{-k}(\pi')$, где π' подобно π, но элементы $(t-k, \ldots, t+k-1, t+k)$ заменены в нем соответственно элементами $(t-k+1, \ldots, t+k, t-k)$. Таким образом, $\mathrm{E}\, A_k + \mathrm{E}\, A_{-k} = 1$.

Пусть $B_k(\pi) =$ [первое добавление на счет и $t+k$, и $t-k$ было $\frac{1}{2}$ и $t+k$ получило второе добавление на счет прежде, чем $t-k$]. Пусть также $C_k(\pi) = [x_{t+k} \ge 2 + A_k]$. Тогда $B_k(\pi) \le C_k(\pi')$, где π' подобно π, но с заменой элементов $(t-k, t-k+1, \ldots, t+k-1)$ элементами $(t+k-1, t-k, \ldots, t+k-2)$. Аналогично $B_{-k}(\pi) \le C_{-k}(\pi'')$, где π'' получено из π посредством замены $(t-k+1, \ldots, t+k-1, t+k)$ элементами $(t-k+2, \ldots, t+k, t-k+1)$. Отсюда следует, что $\mathrm{E}\, B_k \le \mathrm{E}\, C_k$ и $\mathrm{E}\, B_{-k} \le \mathrm{E}\, C_{-k}$.

Доказательство завершается выводом, что $x_{t-k} + x_{t+k} \ge 2 + A_k + A_{-k} - B_k - B_{-k} + C_k + C_{-k}$. [Дальнейшие результаты приводятся в *JACM* **36** (1989), 270–279.]

Верхняя оценка в (17) имеет также соответствующую нижнюю оценку: Эндрю и Фрэнсис Яо доказали, что $\overline{V}_t(n) \ge n + \frac{1}{2}t(\ln\ln n - \ln t - 9)$ при $t > 1$ и $n \ge (8t)^{18t}$ [см. *SICOMP* **11** (1982), 428–447].

26. (a) Обозначим вершины компонентов двух типов через a; $b < c$. Соперник поступает следующим образом при неизбыточных сравнениях. Случай 1, $a : a'$: принять произвольное решение. Случай 2, $x : b$: сказать, что $x > b$; все последующие сравнения $y : b$ с этим конкретным b будут иметь результат $y > b$, в противном случае сравнения выбираются соперником для $U_t(n-1)$, что приводит в общей сложности к $\ge 2 + U_t(n-1)$ сравнениям. Это снижение будем для краткости выражать так: "пусть $b = \min$; $2 + U_t(n-1)$". Случай 3, $x : c$: пусть $c = \max$; $2 + U_{t-1}(n-1)$.

(b) Обозначим новые типы вершин $d_1, d_2 < e$; $f < g < h > i$. Случай 1, $a : a'$ или $c : c'$: произвольное решение. Случай 2, $a : c$: сказать, что $a < c$. Случай 3, $x : b$: пусть $b = \min$; $2 + U_t(n-1)$. Случай 4, $x : d$: пусть $d = \min$; $2 + U_t(n-1)$. Случай 5, $x : e$: пусть $e = \max$;

$3 + U_{t-1}(n-1)$. Случай 6, $x : f$: пусть $f = \min$; $2 + U_t(n-1)$. Случай 7, $x : g$: пусть f и $g = \min$; $3 + U_t(n-2)$. Случай 8, $x : h$: пусть $h = \max$; $3 + U_{t-1}(n-1)$. Случай 9, $x : i$: пусть $i = \min$; $2 + U_t(n-1)$.

(c) Поскольку при $t = 1$ имеем $U_t(n) = n - 1$, в этом случае неравенство выполняется. При $1 < t \le n/2 - 1$ используем индукцию и (a). При $t = n/2$, $U_t(n-1) = U_{t-1}(n-1)$ используем индукцию и (a).

27. (a) Высота h удовлетворяет соотношению $2^h \ge \sum_l 1 \ge \sum_l \Pr(l)/p = 1/p$.

(b) Если $r \le t$, мы достигнем A3 после не менее чем $n - |S_0| - |T_0| = n - |S_0| - r$ подбрасываний монеты. t-й по старшинству элемент будет либо самым малым, либо самым большим в Q, а элементы из Q еще не будут сравниваться один с другим, так что нам понадобится не менее $|Q| - 1$ дополнительных подбрасываний монеты. Если $|S_0| < q$, получим $|Q| = r$, а если нет, то получим $|Q| \ge |S_0| - |C(y_0)| + 1 \ge |S_0| - (q-r) + 1$; так что в обоих случаях будет сделано не менее $n - q$ подбрасываний. Существует $n + 1 - t$ множеств T, в которых содержатся $t - 1$ старших элементов, определяемых данным листом дерева, и для каждого такого T вероятность достичь его равна либо 0, либо $2^{-f}/\binom{n}{t}$, где $f \ge n - q$ есть число подбрасываний монеты, соответствующее T. [Этот соперник был "реализован" в статье Бента (Bent) и Джона (John), *STOC* **17** (1985), 213–216.]

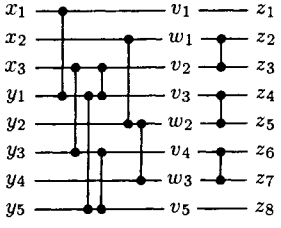
(c) Если $t < r$, замените t значением $n+1-t$; получим, что $t \ge r$, когда r максимизирует правую сторону, поскольку r будет $O(\sqrt{n})$. Если возможно достичь A3 с $|C(y)| > q - r$ при всех $y \in T_0$, алгоритм выполнит $n - 1$ сравнений t-го по старшинству элемента со всеми другими в дополнение не менее чем к $(r-1)(q-r+1)$ сравнениям, которые выполняются между S и $T \setminus \{y_0\}$.

(d) Выберем $r = \lceil \sqrt{m} \rceil$ и $q = 2r - 2$. (Несколько лучше положить $q = r + \lfloor \sqrt{m} + \frac{1}{2} \rfloor - 2$; такой выбор позволит максимизировать нижнюю оценку, выведенную в (c).)

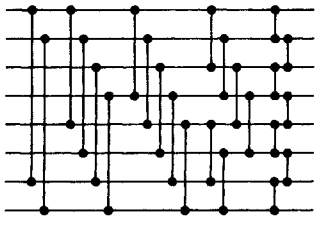
РАЗДЕЛ 5.3.4

1. (Если $m = 2k - 1$ нечетно, то лучше, чтобы в диаграмме за v_k вместо w_{k+1}, v_{k+1}, w_{k+2}, ... следовали элементы v_{k+1}, w_{k+1}, v_{k+2}, ... Такая замена корректна, поскольку линии, которые мы переставляем, сравниваются одна с другим.)

(3,5)-элементное четно-нечетное слияние



8-элементная сортировка Пратта



2. Для приращения h нужно $2 - [2h \ge n]$ уровней; см. приведенную выше диаграмму для $n = 8$.

3. $C(m, m-1) = C(m, m) - 1$ при $m \ge 1$.

4. Если предположить, что $\hat{T}(6) = 4$, то в каждый момент должны действовать три компаратора, так как $\hat{S}(6) = 12$. Но тогда, удалив нижнюю линию и связанные с ней четыре компаратора, мы получили бы $\hat{S}(5) \le 8$, а это — противоречие. [Те же рассуждения показывают, что $\hat{T}(7) = \hat{T}(8) = 6$. Яан Парберри (Ian Parberry) провел длительный компьютерный поиск и пришел к результату $\hat{T}(9) = \hat{T}(10) = 7$; см. *Math. Systems Theory* **24** (1991), 101–116.]

5. Пусть $f(n) = f(\lceil n/2 \rceil) + 1 + \lceil \lg \lceil n/2 \rceil \rceil$, если $n \geq 2$. Тогда, применив индукцию по n, получим $f(n) = (1 + \lceil \lg n \rceil)\lceil \lg n \rceil / 2$.

6. Можно считать, что на каждой стадии выполняется $\lfloor n/2 \rfloor$ сравнений (лишние сравнения не помешают). Так как $\hat{T}(6) = 5$, достаточно доказать, что $T(5) = 5$. В случае $n = 5$ после двух стадий мы не можем избежать частичных упорядочений или , которые нельзя рассортировать за оставшиеся две стадии.

7. Допустим, что исходными ключами будут $\{1, 2, \ldots, 10\}$. Ключевым моментом решения является то, что после первых 16 компараторов линии 2, 3, 4 и 6 не могут содержать ни 8, ни 9, ни 6 и 7 вместе.

8. Это очевидное обобщение теоремы F.

9. $\hat{M}(3,3) \geq \hat{S}(6) - 2\hat{S}(3)$, $\hat{M}(4,4) \geq \hat{S}(8) - 2\hat{S}(4)$, $\hat{M}(5,5) \geq 2\hat{M}(2,3) + 3$, как следует из результатов упр. 8 и $\hat{M}(2,3) \geq \hat{S}(5) - \hat{S}(2) - \hat{S}(3)$. Аналогично $\hat{M}(3,4) = 8$. Но чему равны $\hat{M}(3,5)$ и $\hat{M}(4,5)$?

10. Используйте указание и метод доказательства теоремы Z. Затем покажите, что число нулей в четной подпоследовательности минус число нулей в нечетной подпоследовательности равно ± 1 или 0.

11. (Решение М. У. Грина.) Рассматриваемая сеть симметрична в том смысле, что всякий раз, когда z_i сравнивается с z_j, найдется соответствующее сравнение $z_{2^t-1-j} : z_{2^t-1-i}$. Любая симметричная сеть, способная сортировать последовательность $\langle z_0, \ldots, z_{2^t-1} \rangle$, будет также сортировать последовательность $\langle -z_{2^t-1}, \ldots, -z_0 \rangle$.

Бэтчер заметил, что эта сеть в действительности будет сортировать любой циклический сдвиг $\langle z_j, z_{j+1}, \ldots, z_{2^t-1}, z_0, \ldots, z_{j-1} \rangle$ битонной последовательности. Это следует из принципа нулей и единиц.

[Данный результат *не приложим* к битонным сортировщикам, если их порядок отличен от степени 2. Например, сортировщик на рис. 52 не сможет сортировать последовательность $\langle 0,0,0,0,0,1,0 \rangle$. Сформулированное Бэтчером определение битонной последовательности сложнее и менее удобно, чем то, которым мы пользуемся.]

12. Последовательность $x \vee y$ является битонной (рассмотрите последовательности из 0 и 1), а последовательность $x \wedge y$ — нет (рассмотрите, например, $\langle 3,1,4,5 \rangle \wedge \langle 6,7,8,2 \rangle$).

13. Идеальное тасование заменяет z_i элементом z_j, где двоичное представление j получается из представления i в результате циклического сдвига вправо на один разряд (см. также упр. 3.4.2–13). Рассмотрим перетасовку компараторов, а не линий. Тогда первый столбец компараторов имеет дело с парами $z[i]$ и $z[i \oplus 2^{r-1}]$, следующий столбец — с парами $z[i]$ и $z[i \oplus 2^{r-2}]$, ..., t-й столбец — с $z[i]$ и $z[i \oplus 1]$, $(t+1)$-й столбец — снова с $z[i]$ и $z[i \oplus 2^{r-1}]$ и т. д. Здесь $\oplus$ обозначает операцию "исключающее или" над двоичными представлениями. Эти соображения показывают, что рис. 57 эквивалентен рис. 56; после s стадий мы получаем рассортированные группы из 2^s элементов с чередующимися направлениями упорядочения.

В работе C. G. Plaxton, T. Suel, *Math. Systems Theory* **27** (1994), 491–508, показано, что любая сеть такой структуры имеет, по меньшей мере, $\Omega((\log n)^2/\log\log n)$ уровней задержки.

14. (a) Пусть $y_{i_s} = x_{j_s}$, $y_{j_s} = x_{i_s}$, $y_k = x_k$ при $i_s \neq k \neq j_s$; тогда $y\alpha^s = x\alpha$. (b) Это очевидно до тех пор, пока множество $\{i_s, j_s, i_t, j_t\}$ имеет только три отличающихся элемента; предположим, что $i_s = i_t$. Тогда, если $s < t$, первые $s-1$ компараторов заменят (i_s, j_s, j_t) соответственно элементами (j_s, j_t, i_s) как в $(\alpha^s)^t$, так и в $(\alpha^t)^s$. (c) $(\alpha^s)^s = \alpha$ и $\alpha^1 = \alpha$, так что можно предположить, что $s_1 > s_2 > \cdots > s_k > 1$. (d) Пусть $\beta = \alpha[i:j]$, тогда $g_\beta(x_1, \ldots, x_n) = (\bar{x}_i \vee x_j) \wedge (g_\alpha(x_1, \ldots, x_i, \ldots, x_j, \ldots, x_n) \vee g_\alpha(x_1, \ldots, x_j, \ldots, x_i, \ldots, x_n))$. Построив итеративную процедуру из этих равенств, получим искомый результат. (e) $f_\alpha(x) = 1$

тогда и только тогда, когда не существует пути на графе G_α от i к j при условии $x_i > x_j$. Если α — сеть сортировки, то сопряженная с α сеть также является сетью сортировки и $f_\alpha(x) = 0$ при всех x, удовлетворяющих неравенству $x_i > x_{i+1}$. Примем $x = e^{(i)}$; это показывает, что G не имеет дуг от i к k_1 при некотором $k_1 \neq i$. Если $k_1 \neq i+1$, то $x = e^{(i)} \vee e^{(k_1)}$ показывает, что G имеет дугу от i или k_1 к k_2 при некотором $k_2 \notin \{i, k_1\}$. Если $k_2 \neq i+1$, продолжаем рассуждать так же до тех пор, пока не найдем путь на графе G от i к $i+1$. Обратно, если α не является сетью сортировки, положим, что x — вектор с $x_i > x_{i+1}$ и $g_\alpha(x) = 1$. Для некоторого сопряженного α' получим $f_{\alpha'}(x) = 1$, так что $G_{\alpha'}$ может и не иметь пути от i к $i+1$. [В общем случае $(x\alpha)_i \leq (x\alpha)_j$ при всех x тогда и только тогда, когда $G_{\alpha'}$ имеет ориентированный путь от i к j при всех α', сопряженных с α.]

15. $[1:4][3:2][1:3][2:4][2:3]$.

16. Процесс, очевидно, заканчивается. После каждого выполнения шага T2 выходы с номерами i_q и j_q меняются местами, поэтому в результате выполнения алгоритма сформируется некоторая перестановка выходных линий. Так как получившаяся (стандартная) сеть не изменяет вход $\langle 1, 2, \ldots, n\rangle$, выходные линии должны вернуться на свои прежние места.

17. Сделайте сеть стандартной, воспользовавшись результатом упр. 16. Затем, анализируя входную последовательность $\langle 1, 2, \ldots, n\rangle$, увидим, что стандартные сети выбора должны помещать t наибольших элементов на t линий с наибольшими номерами, а $\hat{V}_t(n)$-сеть должна помещать t-й в порядке убывания элемент на линию $n+1-t$. Примените принцип нулей и единиц.

18. Из доказательства теоремы A следует, что $\hat{V}_t(n) \geq (n-t)\lceil \lg(t+1)\rceil + \lceil \lg t\rceil$.

19. Сеть $[1:n][2:n]\ldots[1:3][2:3]$ выбирает наименьшие два элемента, имея $2n-4$ компараторов. Для $\hat{V}_2(n)$ добавьте $[1:2]$. Нижняя оценка получается из теоремы A (см. ответ к предыдущему упражнению).

20. Прежде всего, заметим, что $\hat{V}_3(n) \geq \hat{V}_3(n-1)+2$, если $n \geq 4$. В силу симметрии можно считать, что первый компаратор есть $[1:n]$, после него расположена сеть для выбора третьего в порядке убывания из $\langle x_2, x_3, \ldots, x_n\rangle$ и еще один компаратор, связанный с линией 1. С другой стороны, $\hat{V}_3(5) \leq 7$, так как 4 компаратора находят минимум и максимум из $\{x_1, x_2, x_3, x_4\}$ и остается рассортировать три элемента.

21. Утверждение ложно. Рассмотрите, например, сети $[1:2][3:4][2:3][1:4][1:2][3:4]$ и $[1:2][3:4][2:3][3:4][1:4][1:2][3:4]$. (Однако Н. Г. де Брейн (N. G. de Bruijn) доказал, что новые компараторы не вносят путаницу в *примитивную* сеть сортировки в смысле упр. 36; см. *Discrete Math.* **9** (1974), 337.)

22. (a) Результат получается индукцией по длине α, поскольку из $x_i \leq y_i$ и $x_j \leq y_j$ следует, что $x_i \wedge x_j \leq y_i \wedge y_j$ и $x_i \vee x_j \leq y_i \vee y_j$. (b) Результат получается индукцией по длине α, поскольку $(x_i \wedge x_j)(y_i \wedge y_j) + (x_i \vee x_j)(y_i \vee y_j) \geq x_i y_i + x_j y_j$. [Следовательно, $\nu(x \wedge y) \leq \nu(x\alpha \wedge y\alpha)$. Автором этого вывода является У. Шокли (W. Shockley).]

23. Пусть $x_k = 1$ тогда и только тогда, когда $p_k \geq j$, $y_k = 1$ тогда и только тогда, когда $p_k > j$. Отсюда $(x\alpha)_k = 1$ тогда и только тогда, когда $(p\alpha)_k \geq j$, и т. д.

24. Формула для l'_i очевидна, а для l'_j выберем $z = x \wedge y$, как в указании. Обратите внимание на то, что из упр. 21 следует, что $(z\alpha)_i = (z\alpha)_j = 0$. Добавление дополнительных единиц к z доказывает существование перестановки p, такой, что $(p\alpha')_j \leq \zeta(z)$, как следует из результатов упр. 23. Соотношения для u'_i и u'_j получаются, если обратить порядок.

25. (Решение Дж. Шапиро (H. Shapiro).) Пусть p и q суть перестановки, такие, что $(p\alpha)_k = l_k$ и $(q\alpha)_k = u_k$. Можно преобразовать p в q за ряд шагов, на каждом из которых

выполняется взаимный обмен пар $(i, i+1)$ соседних целых; такой взаимный обмен приводит к изменению k-го выхода на не более чем ± 1.

26. Существует взаимно однозначное соответствие, которое сопоставляет элементу $\langle p_1, \ldots, p_n \rangle$ из $P_n\alpha$ "последовательность покрытий": $x^{(1)}$ покрывает ... покрывает $x^{(n)}$, где $x^{(i)}$ принадлежат $D_n\alpha$; в этом соответствии $x^{(i-1)} = x^{(i)} \vee e^{(j)}$ тогда и только тогда, когда $p_j = i$. Например, $\langle 3, 1, 4, 2 \rangle$ соответствует такой последовательности: $\langle 1, 1, 1, 1 \rangle$ покрывает $\langle 1, 0, 1, 1 \rangle$ покрывает $\langle 1, 0, 1, 0 \rangle$ покрывает $\langle 0, 0, 1, 0 \rangle$ покрывает $\langle 0, 0, 0, 0 \rangle$. [Эндрю Яо проверил это заключение, протестировав сеть сортировки на $\binom{n}{\lfloor n/2 \rfloor} - 1$ соответствующим образом подобранных перестановках. Например, любая 4-элементная сеть, которая сортирует $\langle 4, 1, 2, 3 \rangle$, $\langle 3, 1, 4, 2 \rangle$, $\langle 3, 4, 1, 2 \rangle$, $\langle 2, 4, 1, 3 \rangle$ и $\langle 2, 3, 4, 1 \rangle$, сортирует любую последовательность. См. упр. 6.5–1; см. также упр. 56.]

27. Этот принцип справедлив, поскольку $(x\alpha)_i$ является i-м по возрастанию элементом в x. Если x и y обозначают различные столбцы матрицы, строки которой упорядочены, т. е. $x_i \le y_i$ при всех i, и если $x\alpha$ и $y\alpha$ обозначают результат сортировки столбцов, то из сформулированного принципа следует, что $(x\alpha)_i \le (y\alpha)_i$ при всех i, поскольку мы можем выбрать i элементов x в тех же строках, в которых находятся данные i элементов y. [Этот принцип был использован для доказательства инвариантного свойства сортировки Шелла (теорема 5.2.1K). Дальнейшее развитие идея получила в интересной статье David Gale, R. M. Karp, *J. Computer and System Sciences* **6** (1972), 103–115. Тот факт, что сортировка столбцов не нарушает упорядоченность строк, был, вероятно, впервые замечен в связи с обработкой таблиц; см. Hermann Boerner, *Darstellung von Gruppen* (Springer, 1955), Chapter V, §5.]

28. Если $\{x_{i_1}, \ldots, x_{i_t}\}$ суть t наибольших элементов, то $x_{i_1} \wedge \ldots \wedge x_{i_t}$ есть t-й элемент. Если $\{x_{i_1}, \ldots, x_{i_t}\}$ *не являются* t-ми наибольшими элементами, то $x_{i_1} \wedge \ldots \wedge x_{i_t}$ *меньше* t-го элемента.

29. $\langle x_1 \wedge y_1, (x_2 \wedge y_1) \vee (x_1 \wedge y_2), (x_3 \wedge y_1) \vee (x_2 \wedge y_2) \vee (x_1 \wedge y_3),\ y_1 \vee (x_3 \wedge y_2) \vee (x_2 \wedge y_3) \vee (x_1 \wedge y_4),\ y_2 \vee (x_3 \wedge y_3) \vee (x_2 \wedge y_4) \vee (x_1 \wedge y_5),\ y_3 \vee (x_3 \wedge y_4) \vee (x_2 \wedge y_5) \vee x_1,\ y_4 \vee (x_3 \wedge y_5) \vee x_2,\ y_5 \vee x_3 \rangle$.

30. Применив законы дистрибутивности и ассоциативности, можно привести любую формулу к набору членов, связанных операцией $\vee$, где каждый член представляет собой объединение посредством операции $\wedge$ исходных переменных; каноническая форма получается затем с помощью законов коммутативности, идемпотентности и поглощения. Далее, S_i — это такие множества S, при которых формула равна 1, если $x_j = [j \in S]$; в то же время формула равна 0, если $x_j = [j \in S']$ для любого собственного подмножества S' множества S.

31. $\delta_4 = 166$. В работе R. Church, *Duke Math. J.* **6** (1940), 732–734, показано, что $\delta_5 = 7579$, а в работе M. Ward, *Bull. Amer. Math. Soc.* **52** (1946), 423, — что $\delta_6 = 7828352$ и следующие значения будут такими: $\delta_7 = 2414682040996$, $\delta_8 = 56130437228687557907786$ [R. Church, *Notices Amer. Math. Soc.* **12** (1965), 724; J. Berman, P. Köhler, *Mitteilungen Math. Seminar Gießen* **121** (1976), 103–124; D. Wiedemann, *Order* **8** (1991), 5–6]. Неизвестно никакой простой формулы для δ_n; в работе D. Kleitman, *Proc. Amer. Math. Soc.* **21** (1969), 677–682, доказано с помощью очень сложных рассуждений, что $(\lg \delta_n)/\binom{n}{\lfloor n/2 \rfloor} \to 1$ при $n \to \infty$.

32. G_{t+1} является также множеством всех цепочек $\theta\psi$, где θ и ψ лежат в G_t и $\theta \le \psi$, как векторы, составленные из 0 и 1. Отсюда следует, что G_t есть множество всех цепочек $z_0 \ldots z_{2^t-1}$ из нулей и единиц, где $z_i \le z_j$, если двоичное представление индекса i "$\le$" двоичного представления j (т. е. оба индекса рассматриваются как векторы из нулей и единиц). Каждый элемент $z_0 \ldots z_{2^t-1}$ множества G_T, кроме $00\ldots0$ и $11\ldots1$, представляет $\wedge$–$\vee$-функцию $f(x_1, \ldots, x_t)$ из D_{2^t} при соответствии $f(x_1, \ldots, x_t) = z[(x_1 \ldots x_t)_2]$.

33. Если бы такая сеть существовала, то мы получили бы, что $(x_1 \wedge x_2) \vee (x_2 \wedge x_3) \vee (x_3 \wedge x_4) = f(x_1 \wedge x_2, x_1 \vee x_2, x_3, x_4)$ или $f(x_1 \wedge x_3, x_2, x_1 \vee x_3, x_4)$ или ... или $f(x_1, x_2, x_3 \wedge x_4, x_3 \vee x_4)$

для некоторой функции f. Подставляя $\langle x_1, x_2, x_3, x_4\rangle = \langle x, \bar{x}, 1, 0\rangle$, $\langle x, 0, \bar{x}, 1\rangle$, $\langle x, 1, 0, \bar{x}\rangle$, $\langle 1, x, \bar{x}, 0\rangle$, $\langle 1, x, 0, \bar{x}\rangle$, $\langle 0, 1, x, \bar{x}\rangle$, находим, что такой функции f не существует.

34. Да; доказав это, вы можете взяться за сеть, представленную на рис. 49, при $n = 16$ (если только вы просто не проверите это на всех двоичных векторах 2^n, воспользовавшись теоремой Z).

35. В противном случае перестановка, в которой только i и $i+1$ находятся не на своих местах, никогда не была бы рассортирована. Пусть D_k — номера компараторов $[i{:}i{+}k]$ в стандартной сети сортировки. Тогда $D_1 + 2D_2 + D_3 \geq 2(n-2)$, поскольку должно существовать два компаратора от $\{i, i{+}1\}$ до $\{i{+}2, i{+}3\}$ при $1 \leq i \leq n-3$ так же, как $[1{:}2]$ и $[n{-}1{:}n]$. Аналогично $D_1 + 2D_2 + \cdots + kD_k + (k-1)D_{k-1} + \cdots + D_{2k-1} \geq k(n-k)$. Эта формула предложена Дж. М. Поллардом (J. M. Pollard). Можно также доказать, что $2D_1 + D_2 \geq 3n - 4$. Если удалить первые компараторы вида $[j{:}j{+}1]$ для всех j, то должен существовать, по меньшей мере, еще один компаратор, размещенный в пределах $\{i, i{+}1, i{+}2\}$ при $1 \leq i \leq n{-}2$. Аналогично $kD_1 + (k-1)D_2 + \cdots + D_k \geq S(k+1)(n-k) + k(k-1)$.

36. (a) Каждое сравнение соседних элементов уменьшает число инверсий на 0 или 1, а $\langle n, n{-}1, \ldots, 1\rangle$ имеет $\binom{n}{2}$ инверсий. (b) Пусть $\alpha = \beta[p{:}p{+}1]$ и будем рассуждать по индукции по длине α. Если $p = i$, то $j > p+1$, и $(x\beta)_p > (x\beta)_j$, $(x\beta)_{p+1} > (x\beta)_j$; следовательно, $(y\beta)_p > (y\beta)_j$ и $(y\beta)_{p+1} > (y\beta)_j$. Если $p = i-1$, то либо $(x\beta)_p$, либо $(x\beta)_{p+1} > (x\beta)_j$; следовательно, либо $(y\beta)_p$, либо $(y\beta)_{p+1} > (y\beta)_j$. Если $p = j-1$ или j, рассуждения аналогичны. Для остальных p доказательство тривиально.

Замечание. Если α является сетью сортировки, то α^R — тоже сеть сортировки (в ней компараторы расположены в обратном порядке). Более общий случай и другое доказательство (c) приведены в работе N. G. de Bruijn, *Discrete Mathematics* **9** (1974), 333–339; *Indagationes Math.* **45** (1983), 125–132. В этой статье доказано, что примитивная сеть сортирует все перестановки мультимножества $\{n_1 \cdot 1, \ldots, n_m \cdot m\}$ тогда и только тогда, когда она сортирует единственную перестановку $m^{c_m} \ldots 1^{c_1}$. Отношение $x \preceq y$, определено для перестановок x и y таким образом, что означает существование стандартной сети α, такой, что $x = y\alpha$, и называется *порядком Брюа*; аналогичное отношение, но ограниченное только примитивом α, есть *слабый порядок Брюа* (см. ответ к упр. 5.2.1–44).

37. Достаточно показать, что если заменить каждый компаратор операцией *взаимной перестановки*, то получится отражающая сеть (reflection network), преобразующая $\langle x_1, \ldots, x_n\rangle$ в $\langle x_n, \ldots, x_1\rangle$. Но при такой интерпретации нетрудно проследить путь x_k. Обратите внимание на то, что перестановка $\pi = (1\ 2)(3\ 4)\ldots(2n{-}1\ 2n)(2\ 3)(4\ 5)\ldots(2n{-}2\ 2n{-}1) = (1\ 3\ 5\ \ldots\ 2n{-}1\ 2n\ 2n{-}2\ \ldots\ 2)$ удовлетворяет условию $\pi^n = (1\ 2n)(2\ 2n{-}1)\ldots(n{-}1\ n)$. Четно-нечетная сортировка с транспозициями была вскользь упомянута X. Сьювордом (H. Seward) в 1954 году; она была проанализирована в работах A. Grasselli, *IRE Trans.* **EC-11** (1962), 483, и Kautz, Levitt, Waksman, *IEEE Trans.* **C-17** (1968), 443–451. Свойство отражения такой сети было придумано значительно раньше Г. Э. Дьюдени (H. E. Dudeney) в одной из его головоломок [см. *Amusements in Mathematics* (1917), 193].

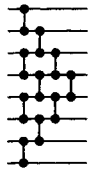
38. Вставьте элементы $i_1, \ldots, i_N$ в первоначально пустую диаграмму, воспользовавшись алгоритмом 5.1.4I, но выполнив одно весьма существенное изменение: установите $P_{ij} \leftarrow x_i$ на шаге I3 только в том случае, если $x_i \neq P_{i(j-1)}$. Можно доказать, что x_i будет равно $P_{i(j-1)}$ на этом шаге, только если $x_i + 1 = P_{ij}$, когда входы $i_1 \ldots i_N$ определяют примитивную сеть сортировки. (Заключенный в скобки комментарий придется соответственно модифицировать.) После того как i_j будет вставлено в P, установите $Q_{st} \leftarrow j$, как в теореме 5.1.4A. После N шагов диаграмма P всегда будет содержать $(r, r+1, \ldots, n-1)$ в строке r, в то время как Q будет представлять собой диаграмму, из которой можно восстановить последовательность $i_1 \ldots i_N$, выполняя операции в обратном порядке.

Например, если $n = 6$, то последовательность $i_1 \ldots i_N = 4\,1\,3\,2\,4\,3\,5\,4\,3\,1\,2\,3\,5\,1\,4$ соответствует

$$P = \begin{array}{|c|c|c|c|c|}\hline 1 & 2 & 3 & 4 & 5 \\ \hline 2 & 3 & 4 & 5 \\ \cline{1-4} 3 & 4 & 5 \\ \cline{1-3} 4 & 5 \\ \cline{1-2} 5 \\ \cline{1-1}\end{array}\,, \qquad Q = \begin{array}{|c|c|c|c|c|}\hline 1 & 4 & 5 & 8 & 13 \\ \hline 2 & 6 & 7 & 15 \\ \cline{1-4} 3 & 9 & 12 \\ \cline{1-3} 10 & 11 \\ \cline{1-2} 14 \\ \cline{1-1}\end{array}\,.$$

Транспозиция Q соответствует дополняющей сети $[n-i_1 : n-i_1+1] \ldots [n-i_N : n-i_N+1]$.

[*См.* A. Lascoux, M. P. Schützenberger, *Comptes Rendus Acad. Sci. Paris* (I) **295** (1982), 629–633; R. P. Stanley, *Eur. J. Combinatorics* **5** (1984), 359–372; P. H. Edelman, C. Greene, *Advances in Math.* **63** (1987), 42–99.] Диаграммы примитивных сетей сортировки также соответствуют организации псевдолиний и других абстракций при двумерном представлении сложности сетей. Более подробно об этом речь идет в работе D. E. Knuth, *Lecture Notes in Comp. Sci.* **606** (1992).



39. Если, например, $n = 8$, то такая сеть должна включать показанные здесь компараторы. Все другие компараторы не будут задействованы в наборе 10101010. Тогда линии $\lceil n/3 \rceil .. \lceil 2n/3 \rceil = 3..6$ сортируют 4 элемента, как и в упр. 37. (Это упражнение базируется на идее Дэвида Б. Уилсона (David B. Wilson).)

Замечание. Существует взаимно однозначное соответствие между примитивной сетью минимальной длины, которая сортирует данную цепочку битов, и диаграммами Юнга, вид которых ограничен зигзагообразным путем, определенным этой цепочкой битов. Таким образом, из упр. 38 следует взаимно однозначное соответствие между примитивной сетью из $\binom{n/2+1}{2}$ компараторов, которая сортирует $(10)^{n/2}$, и примитивной сетью из $\binom{n/2+1}{2}$ компараторов, которая сортирует $n/2 + 1$ произвольных чисел. Если примитивная сеть сортирует цепочку битов $1^{n/2}0^{n/2}$, можно сделать следующий вывод: все ее "половинки", состоящие из подсетей с линиями от k до $k + n/2$ включительно, также являются сетями сортировки при $1 \le k \le n/2$. (См. также теорему де Брейна, которая цитируется в ответе к упр. 36.)

40. Это вытекает из применения неравенств относительно конечных членов интересующего нас построения, которые представлены в п. 7 статьи H. Rost, *Zeitschrift für Wahrscheinlichkeitstheorie und verwandte Gebiete* **58** (1981), 41–53. В ней положено $b = \frac{1}{2}$, $a = \frac{1}{4}$ и $t = 4n + \sqrt{n} \ln n$.

Эксперименты показали, что ожидаемое время достижения *любой* примитивной сети сортировки — не обязательно по методу пузырька — очень близко к $2n^2$. Кстати, Р. П. Стэнли (R. P. Stanley) и С. В. Фомин доказали, что если компараторы $[i_k : i_k+1]$ выбираются не с равной вероятностью, а таким образом, что $i_k = j$ возникает с вероятностью $j/\binom{n}{2}$, то соответствующее ожидаемое время становится точно равным $\binom{n}{2} H_{\binom{n}{2}}$.

42. Должен существовать путь длиной $\lceil \lg n \rceil$ или больше из некоторого входа в наибольший выход (рассмотрите m_n в теореме A). Если поместить на этот вход ∞, то поведение всех компараторов на этом пути будет предопределено, а оставшаяся сеть должна быть $(n-1)$-сортировщиком. [*IEEE Trans. on Computers* **C-21** (1972), 612–613.]

45. После l уровней вход x_1 может оказаться в не более чем 2^t разных местах. После завершения слияния x_1 может оказаться в $n + 1$ разных местах.

46. [См. *J. Algorithms* **3** (1982), 79–88; приведенное ниже доказательство предложено В. С. Гринбергом.] Можно предположить, что $1 \le m \le n$ и что на любой стадии выполняется m сравнений. Пусть $l = \lceil (n-m)/2 \rceil$, и предположим, нужно выполнить слияние $x_1 \le \cdots \le x_m$ с $y_1 \le \cdots \le y_n$. Противник может принудить выполнить $\lceil \lg(m+n) \rceil$ стадий следующим образом: на первой стадии некоторое x_j сравнивается с элементом y_k, причем

либо $k \le l$, либо $k \ge l+m$. Противник решает, что $x_{j-1} < y_1$ и $x_{j+1} > y_n$, а также, что $x_j > y_k$, если $k \le l$, и $x_j < y_k$, если $k \ge l+m$. Остальная часть задачи — это, по сути, слияние x_j либо с $y_{k+1} \le \cdots \le y_n$, либо с $y_1 \le \cdots \le y_{k-1}$. Таким образом, остается, по крайней мере, $\min(n-k+1, k) \ge \min(n-l+1, l+m) = \lceil (m+n)/2 \rceil$ исходов. Отсюда следует, что необходимо не менее чем $\lceil \lg \lceil (m+n)/2 \rceil \rceil = \lceil \lg(m+n) \rceil - 1$ последующих стадий.

48. Пусть u — наименьший элемент среди $(x\alpha)_j$, а $y^{(0)}$ — любой вектор из множества D_n, такой, что из $(y^{(0)})_k = 0$ следует, что $(x\alpha)_k$ содержит элемент $\le u$, а из $(y^{(0)})_k = 1$ следует, что $(x\alpha)_k$ содержит элемент $> u$. Если $\alpha = \beta[p:q]$, то можно найти вектор $y^{(1)}$, удовлетворяющий тому же условию, в котором α заменено элементом β, и такой, что $y^{(1)}[p:q] = y^{(0)}$. Начав с $(y^{(0)})_i = 1$, $(y^{(0)})_j = 0$, мы, в конце концов, получим вектор $y = y^{(r)}$, удовлетворяющий требуемому условию.

Ж. Боде (G. Baudet) и Д. Стивенсон (D. Stevenson) заметили, что из комбинации результатов упр. 37 и 48 следует простой метод параллельной сортировки с $(n \ln n)/k + O(n)$ циклами сравнения на k процессорах. Для этого следует сначала рассортировать k подфайлов размером $\le \lceil n/k \rceil$, а затем слить их за k проходов, используя "четно-нечетное слияние с транспозициями" порядка k. [*IEEE Trans.* **C-27** (1978), 84–87.]

49. Как $(x \vee\!\!\!\vee y) \vee\!\!\!\vee z$, так и $x \vee\!\!\!\vee (y \vee\!\!\!\vee z)$ представляют собой наибольшие m элементов мультимножества $x \uplus y \uplus z$; $(x \wedge\!\!\!\wedge y) \wedge\!\!\!\wedge z$ и $x \wedge\!\!\!\wedge (y \wedge\!\!\!\wedge z)$ представляют собой наименьшие m элементов. Если $x = y = z = \{0,1\}$, то $(x \wedge\!\!\!\wedge z) \vee\!\!\!\vee (y \wedge\!\!\!\wedge z) = (x \wedge\!\!\!\wedge y) \vee\!\!\!\vee (x \wedge\!\!\!\wedge z) \vee\!\!\!\vee (y \wedge\!\!\!\wedge z) = \{0,0\}$, в то время как средние элементы в $\{0,0,0,1,1,1\}$ суть $\{0,1\}$. Из анализа сети сортировки для трех элементов и результатов упр. 48 следует, что средние элементы $x \uplus y \uplus z$ могут быть выражены либо как $((x \vee\!\!\!\vee y) \wedge\!\!\!\wedge z) \vee\!\!\!\vee (x \wedge\!\!\!\wedge y)$, либо как $((x \wedge\!\!\!\wedge y) \vee\!\!\!\vee z) \wedge\!\!\!\wedge (x \vee\!\!\!\vee y)$, либо с помощью любой другой формулы, получаемой путем перестановки переменных x, y, z в этих выражениях. (По-видимому, для средних элементов симметричной формулы не существует.)

50. Точно так, как в теореме Z, можно найти все тождества, удовлетворяемые операцией

$$x \vee\!\!\!\vee y = \min(x+y, 1), \qquad x \wedge\!\!\!\wedge y = \max(0, x+y-1)$$

на множестве рациональных чисел x, y в диапазоне $[0\,..\,1]$. [Это операция "переливания" как можно большего количества жидкости из одного стакана, в который налиты x, в другой, в который ранее были налиты y, что подмечено Дж. М. Поллардом (J. M. Pollard).] Все такого рода тождества можно получить из системы четырех аксиом и правила интерференции для многозначной логики Лукашевича (Łukasiewicz); см. также работу Rose, Rosser, *Trans. Amer. Math. Soc.* **87** (1958), 1–53.

51. Пусть $\alpha' = \alpha[i:j]$, а k — произвольный индекс $\ne i, j$. Если $(x\alpha)_i \le (x\alpha)_k$ при всех x, то $(x\alpha')_i \le (x\alpha')_k$; если $(x\alpha)_k \le (x\alpha)_i$ и $(x\alpha)_k \le (x\alpha)_j$ при всех x, тогда то же самое имеет место, если заменить α элементом α'; если $(x\alpha)_k \le (x\alpha)_i$ при всех x, то $(x\alpha')_k \le (x\alpha')_j$. Таким образом, мы видим, что для α' существует, по крайней мере, столько же соотношений, сколько для α, плюс еще одно, если $[i:j]$ не является избыточным. [*Bell System Tech. J.* **49** (1970), 1627–1644.]

52. (a) Будем рассматривать сортировку нулей и единиц. Пусть $w = x_0 + x_1 + \cdots + x_N$. Сеть не выполнит работу тогда и только тогда, когда $w \le t$ и $x_0 = 1$, прежде чем завершится N-сортировка. Если в этот момент $x_0 = 1$, то в начале должна была быть единица и при $1 \le j \le n$ мы должны были в начале иметь либо $x_{2j-1+2nk} = 1$ при $0 \le k \le m$, либо $x_{2j+2nk} = 1$ при $0 \le k \le m$; таким образом, $w \ge 1 + (m+1)n = t$. Итак, отказ свидетельствует о том, что $w = t$ и $x_j = x_{j+2nk}$ при $1 \le k \le m$ и $x_{2j} = \bar{x}_{2j-1}$ при $1 \le j \le n$. Более того, специальная подсеть должна преобразовать эти входы таким образом, что $x_{2m+2n+j} = 1$ при $1 \le j \le m$.

(b) Например, специальная подсеть для $(y_1 \vee y_2 \vee \bar{y}_3) \wedge (\bar{y}_2 \vee y_3 \vee \bar{y}_4) \wedge \ldots$ могла бы быть

$$[1+2n{:}2mn+2n+1][3+2n{:}2mn+2n+1][6+2n{:}2mn+2n+1]$$
$$[4+4n{:}2mn+2n+2][5+4n{:}2mn+2n+2][8+4n{:}2mn+2n+2]\ldots,$$

если использовать $x_{2j-1+2kn}$ и x_{2j+2kn} для представления y_j и $\bar{y}_j$ в k-м подвыражении и $x_{2m+2n+k}$ — для представления самого этого выражения.

53. Раскрасьте все линии красным или синим цветом в соответствии со следующим правилом:

если $i \bmod 4$ равно	тогда цвет линии i в случае (a)	цвет в случае (b)
0	красный	красный
1	синий	красный
2	синий	синий
3	красный	синий

Теперь вы увидите, что первые $t-1$ уровней сети состоят из двух отдельных сетей: одна из 2^{t-1} красных линий, а другая — из 2^{t-1} синих линий. Компараторы на t-м уровне образуют сеть слияния, как в сетях битонного или четно-нечетного слияния. Таким образом, мы получили искомый результат при $k=1$.

Красно-синяя декомпозиция также годится и для случая $k=2$. Если вход 4-упорядочен, красные линии содержат 2^{t-1} чисел, которые являются 2-упорядоченными; то же самое можно сказать относительно синих линий. Так что мы оказались в ситуации с

$$x_0y_0y_1x_1x_2y_2y_3x_3\ldots \text{ (случай (a))} \qquad \text{или} \qquad x_0x_1y_0y_1x_2x_3y_2y_3\ldots \text{ (случай (b))}$$

после $t-1$ уровней; окончательный результат

$$(x_0\wedge y_0)(x_0\vee y_0)(y_1\wedge x_1)(y_1\vee x_1)\ldots \qquad \text{или} \qquad x_0(x_1\wedge y_0)(x_1\vee y_0)(y_1\wedge x_2)(y_1\vee x_2)\ldots$$

совершенно очевидно является 2-упорядоченным.

Теперь для $k \geq 2$ можно предположить, что $k \leq t$. Первые $t-k+2$ уровней разделяются в результате декомпозиции на 2^{k-2} отдельных сетей размером 2^{t-k+2}, каждая из которых является 2-упорядоченной в случае $k=2$; следовательно, линии являются 2^{k-1}-упорядоченными после $t-k+2$ уровней. Последующие уровни, очевидно, сохраняют 2^{k-1}-упорядочение, поскольку они обладают "вертикальной" периодичностью порядка 2^{k-2}. (Можно представить себе $-\infty$ на линиях $-1, -2, \ldots$ и $+\infty$ — на линиях $2^t, 2^t+1, \ldots$.)

Литература. Сеть (a) впервые была рассмотрена в работе M. Dowd, Y. Perl, L. Rudolph, M. Saks, *JACM* **36** (1989), 738–757; сеть (b) — в работе E. R. Canfield, S. G. Williamson, *Linear and Multilinear Algebra* **29** (1991), 43–51. Интересно отметить, что в случае (a) имеем $D_n\alpha = G_t$, где G_t определено в упр. 32 [Дауд и др., теорема 17]; таким образом, изображения D_n самого по себе недостаточно для того, чтобы охарактеризовать поведение периодической сети.

54. Следующее построение выполнено в работе Ajtai, Komlós, Szemerédi [*FOCS* **33** (1992), 686–692]. Оно показывает, как рассортировать m^3 элементов с четырьмя уровнями m^2-элементных сортировщиков: предположим, что сортируемые элементы суть нули и единицы; пронумеруем линии $(a,b,c) = am^2+bm+c$ при $0 \leq a,b,c < m$. Первый уровень сортирует линии $\{(a,b,(b+k) \bmod m) \mid 0 \leq a,b < m\}$ при $0 \leq k < m$; пусть a_k — число единиц в k-й группе из m^2 линий. Второй уровень сортирует $\{(a,b,k) \mid 0 \leq a,b < m\}$ при $0 \leq k < m$; число единиц в k-й группе тогда будет таким:

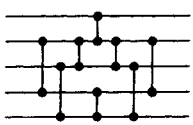
$$b_k = \sum_{j=0}^{m^2-1} \left\lfloor \frac{a_{(k-j) \bmod m} + j}{m^2} \right\rfloor,$$

и отсюда следует, что $b_0 \le b_1 + 1$, $b_1 \le b_2 + 1$, $\ldots$, $b_{m-1} \le b_0 + 1$. На третьем уровне сортируем $\{(k,a,b) \mid 0 \le a,b < m\}$ при $0 \le k < m$; число единиц в k-й группе тогда будет таким:

$$c_k = \sum_{i=0}^{m-1} \sum_{j=0}^{m-1} \left\lfloor \frac{b_i + km + j}{m^2} \right\rfloor .$$

Если $0 < c_{k+1} < m^2$, имеем $c_k \le \binom{m-1}{2}$ и $c_j = 0$ при $j < k$. Аналогично, если $0 < c_k < m^2$, имеем $c_{k+1} \ge m^2 - \binom{m-1}{2}$ и $c_j = 0$ при $j > k+1$. Следовательно, четвертый уровень, который сортирует линии $m^2k - \binom{m-1}{2} \mathinner{..} m^2k + \binom{m-1}{2} - 1$ при $0 < k < m$, завершит сортировку.

Из всего сказанного следует, что четыре уровня из m-элементных сортировщиков рассортируют $f(m) = \lfloor \sqrt{m} \rfloor^3$ элементов, а 16 уровней рассортируют $f(f(m))$ элементов. Это доказывает исходное утверждение, поскольку $f(f(m)) > m^2$, когда $m > 24$. (Описанное построение не относится к "компактным", так что, возможно, вам удастся достичь того же результата и с меньшим числом уровней.)

55. 

[Если $P(n)$ обозначает минимальное число переключателей, необходимых в перестановочной сети, то ясно, что $P(n) \ge \lceil \lg n! \rceil$. Несколько изменив построение, как предлагается в работе L. J. Goldstein, S. W. Leibholz, *IEEE Trans.* **EC-16** (1967), 637–641, можно показать, что $P(n) \le P(\lfloor n/2 \rfloor) + P(\lceil n/2 \rceil) + n - 1$. Следовательно, $P(n) \le B(n)$ для всех n, где $B(n)$ есть функция двоичной вставки из 5.3.1–(3). М. У. Грин (M. W. Green) показал, что $P(5) = 8$.]

56. Можно построить α_x по индукции так, что $x\alpha_x = 0^{k-1}101^{n-k-1}$, если x имеет k нулей. Основной случай, α_{10} — вырожденный. Иначе применяется, по крайней мере, один из следующих четырех случаев, где y не рассортировано: (1) $x = y0$, $\alpha_x = \alpha_y[n{-}1{:}n][n{-}2{:}n{-}1]\ldots[1{:}2]$. (2) $x = y1$, $\alpha_x = \alpha_y[1{:}n][2{:}n]\ldots[n{-}1{:}n]$. (3) $x = 0y$, $\alpha_x = \alpha_y^+[1{:}n][1{:}n{-}1]\ldots[1{:}2]$. (4) $x = 1y$, $\alpha_x = \alpha_y^+[1{:}2][2{:}3]\ldots[n{-}1{:}n]$. Сеть α^+ получается из α в результате замены каждого компаратора $[i{:}j]$ компаратором $[i{+}1{:}j{+}1]$. [См. M. J. Chung, B. Ravikumar, *Discrete Math.* **81** (1990), 1–9.] В этой конструкции используется $\binom{n}{2} - 1$ компараторов; а можно ли достичь того же результата с существенно меньшим числом компараторов?

57. [См. H. Zhu, R. Sedgewick, *STOC* **14** (1982), 296–302.] Указанное время задержки легко проверяется по индукции, но проблема анализа рекуррентного соотношения

$$A(m,n) = A(\lfloor m/2 \rfloor, \lceil n/2 \rceil) + A(\lceil m/2 \rceil, \lfloor n/2 \rfloor) + \lceil m/2 \rceil + \lceil n/2 \rceil - 1,$$

когда $A(0,n) = A(m,0) = 0$, более сложна.

В процессе битонного слияния выполняется $B(m,n) = C'(m+n)$ сравнений (см. (15)). Таким образом, можно использовать то, что $\{\lfloor m/2 \rfloor + \lceil n/2 \rceil, \lceil m/2 \rceil + \lfloor n/2 \rfloor\} = \{\lfloor (m+n)/2 \rfloor, \lceil (m+n)/2 \rceil\}$, чтобы показать, что $B(m,n) = B(\lfloor m/2 \rfloor, \lceil n/2 \rceil) + B(\lceil m/2 \rceil, \lfloor n/2 \rfloor) + \lfloor (m+n)/2 \rfloor$. Тогда по индукции $A(m,n) \le B(m,n)$.

Пусть $D(m,n) = C(m+1,n+1) + C(m,n) - C(m+1,n) - C(m,n+1)$. Имеем $D(0,n) = D(m,0) = 1$ и $D(m,n) = 1$, когда $m+n$ нечетно. В противном случае $m+n$ четно, $mn \ge 1$ и имеем $D(m,n) = D(\lfloor m/2 \rfloor, \lfloor n/2 \rfloor) - 1$. Следовательно, $D(m,n) \le 1$ при всех $m,n \ge 0$.

Рекуррентное соотношение для A эквивалентно рекуррентному соотношению для C, за исключением случая, когда оба параметра — и m, и n — нечетны. Но и в этом случае имеем $A(m,n) \ge C(\lfloor m/2 \rfloor, \lceil n/2 \rceil) + C(\lceil m/2 \rceil, \lfloor n/2 \rfloor) + \lceil m/2 \rceil + \lceil n/2 \rceil - 1 = C(m,n) + 1 - D(\lfloor m/2 \rfloor, \lfloor n/2 \rfloor) \ge C(m,n)$ по индукции.

Пусть $l = \lceil \lg \min(m,n) \rceil$. На уровне k четно-нечетной рекурсии при $0 \le k < l$ выполняем 2^k слияний соответствующих размеров $(m_{jk}, n_{jk}) = (\lfloor (m+j)/2^k \rfloor, \lfloor (n + 2^k - 1 - j)/2^k \rfloor)$ при $0 \le j < 2^k$. Цена рекурсии, $\sum_j (\lceil m_{jk}/2 \rceil + \lceil n_{jk}/2 \rceil - 1)$, есть $f_k(m) + f_k(n) - 2^k$; можно

записать $f_k(n) = \max(n'_k, n - n'_k)$, где $n'_k = 2^k\lfloor n/2^{k+1} + 1/2\rfloor$ кратно 2^k, ближайшему к $n/2$. Поскольку $0 \le f_k(n) - n/2 \le 2^{k-1}$, суммарная цена рекурсии для уровней от 0 до $l-1$ лежит между $\frac{1}{2}(m+n)l - 2^l$ и $\frac{1}{2}(m+n)l$.

Наконец, если $m \le n$, то 2^l слияний (m_{jl}, n_{jl}) на уровне l имеют $m_{jl} = 0$ при $0 \le j < 2^l - m$ и $m_{jl} = 1$ — при других значениях m из j. Поскольку $A(1,n) = n$, суммарная цена уровня l равна $\sum_{k=n}^{m+n-1}\lfloor k/2^l\rfloor \le \sum_{k=n}^{m+n-1} k/m = \frac{m-1}{2} + n$.

Таким образом, четно-нечетное слияние, в отличие от битонного слияния, характеризуется $O(m+n)$ оптимального числа сравнений $\hat{M}(m,n)$. Наши рассуждения показывают фактически, что $A(m,n) = \sum_{k=0}^{l-1}(f_k(m) + f_k(n) - 2^k) + g_l(m+n) - g_l(\max(m,n))$, где $g_l(n)$ можно представить в форме $\sum_{k=0}^{n-1}\lfloor k/2^l\rfloor = \lfloor n/2^l\rfloor(n - 2^{l-1}(\lfloor n/2^l\rfloor + 1))$.

58. Если $h[k+1] = h[k] + 1$ и файл еще не упорядочен, то на следующем проходе с ним обязательно произойдут какие-нибудь изменения. Как показано в упр. 5.2.2–1, число инверсий уменьшится и, следовательно, файл, в конце концов, рассортируется. Но если $h[k+1] \ge h[k] + 2$ при $1 \le k < m$, то наименьший ключ никогда не попадает в нужное место, если он первоначально находится в R_2.

59. Используем указание и рассмотрим следующий случай: $K_{N+1} = K_{N+2} = \cdots = 1$. Если $K_{h[1]+j} = \cdots = K_{h[m]+j} = 1$ на шаге j и если $K_i = 0$ при некотором $i > h[1] + j$, то должно оказаться $i < h[m] + j$, так как число единиц не превышает n. Предположим, что k и i минимальные, такие, что $h[k] + j < i < h[k+1] + j$ и $K_i = 0$. Пусть $s = h[k+1] + j - i$; имеем $s < h[k+1] - h[k] \le k$. На шаге $j - s$ под головками должно быть, по крайней мере, $k+1$ нулей, так как $K_i = K_{h[k+1]+j-s}$ устанавливается равным нулю на этом шаге; еще через s шагов между $K_{h[1]+j}$ и K_i включительно остается не менее $k+1-s \ge 2$ нулей, что противоречит минимальности i.

При втором проходе следующие $n-1$ элементов помещаются на свои места и т. д. Если мы начинаем с перестановки $N\ N{-}1\ \ldots\ 2\ 1$, то при первом проходе она принимает вид

$$N{+}1{-}n\ N{-}n\ \ldots\ 1\ N{+}2{-}n\ \ldots\ N{-}1\ N,$$

поскольку $K_{h[1]+j} > K_{h[m]+j}$ всякий раз, когда $1 \le h[1] + j$ и $h[m] + j \le N$; следовательно, приведенная оценка является наилучшей из возможных.

60. Предположим, что $h[k+1] - s > h[k]$ и $h[k] \le s$; если наименьший ключ вначале находился в позиции R_{n-s}, то, в конце концов, он окажется в позиции R_i при $i > 1$. Следовательно, условие $h[k+1] \le 2h[k]$ является необходимым; оно также является достаточным в частном случае, при $t = 0$, следующей теоремы.

Теорема. *Если $n = N$, а $K_1 \ldots K_N$ — перестановка множества $\{1, 2, \ldots, n\}$, то при одном проходе сортировки будет установлено $K_i = i$ при $1 \le i \le t{+}1$, если $h[k{+}1] \le h[k]{+}h[k{-}i]{+}i$ при $1 \le k < m$ и $0 \le i \le t$.* (Условимся, что $h[k] = k$, если $k \le 0$.)

Доказательство. Применяем индукцию по t. Если на шаге t ключ $t+1$ не находится под головками, то можно считать, что он расположен в позиции $R_{h[k+1]+t-s}$ при некотором $s > 0$, где $h[k+1] - s < h[k]$; следовательно, $h[k-t] + t - s > 0$. Но это невозможно, если рассмотреть шаг $t - s$, на котором, вероятно, элемент $t+1$ был помещен в позицию $R_{h[k+1]+t-s}$, хотя имелись, по крайней мере, $t+1$ меньших активных головок. ▌

(Это условие необходимо при $t = 0, 1$, но не при $t = 2$.)

61. Если сортируются числа $\{1, \ldots, 23\}$, то в соответствии с теоремой из предыдущего упражнения получаем, что $\{1, 2, 3, 4\}$ попадают на свои окончательные места. Можно проверить, что в случае сортировки нулей и единиц на шагах -2, -1 и 0 невозможно такое положение, когда все головки читают 0, а во всех позициях не под головками содержится 1; следовательно, доказательство в предыдущем упражнении можно расширить и, таким

образом, показать, что $\{5, 6, 7\}$ также попадают на свои места. Наконец, то, что $\{8, \ldots, 23\}$ должны сортироваться, следует из рассуждений, приведенных в упр. 59.

63. Если $r \le m - 2$, то головки преобразуют цепочку $0^p1^101^301^70\ldots01^{2^r-1}01^q$ в цепочку $0^{p+1}1^101^301^70\ldots01^{2^{r-1}-1}01^{2^r-1+q}$; следовательно, необходимо $m - 2$ проходов. [Для случая, когда головки находятся в позициях $1, 2, 3, 5, \ldots, 1 + 2^{m-2}$, Пратт получил аналогичный результат: цепочка $0^p1^a01^{2^b-1}01^{2^{b+1}-1}0\ldots01^{2^r-1}01^q$, $1 \le a \le 2^{b-1}$, превращается в $0^{p+1}1^{a-1}01^{2^b-1}01^{2^{b+1}-1}0\ldots01^{2^{r-1}-1}01^{2^r+q}$; значит, для такой последовательности головок в наихудшем случае требуется не менее $m - \lceil \log_2 m \rceil - 1$ проходов. Эта последовательность головок особенно интересна, поскольку она была использована как основа весьма остроумного сортирующего устройства, изобретенного Ф. Н. Армстронгом (P. N. Armstrong) [см. *U. S. Patent 3399383* (1965)]. Пратт предполагает, что эти исходные последовательности представляют наихудший случай из возможных.]

64. При быстрой сортировке каждый ключ $K_2, \ldots, K_N$ сравнивается с K_1; пусть $A = \{i \mid K_i < K_1\}$, $B = \{j \mid K_j > K_1\}$. Далее метод быстрой сортировки независимо применяется к A и B. Все сравнения $K_i : K_j$ для i в A и j в B запрещаются как при быстрой сортировке, так и в алгоритме ограниченной однородной сортировки, но никакие другие сравнения не запрещаются в алгоритме неограниченной однородной сортировки.

В этом случае мы могли бы еще больше ограничить алгоритм, опуская случаи 1 и 2 так, что к G добавляются дуги, только если явно выполняются сравнения, и рассматривая при проверке избыточности лишь пути длиной 2. Другой способ решения этой задачи заключается в том, чтобы рассмотреть эквивалентный алгоритм вставки в дерево (раздел 6.2.2), который выполняет те же сравнения, что и однородный алгоритм, и в том же порядке.

65. Вероятность того, что K_{a_i} сравнивается с K_{b_i}, — это вероятность того, что c_i других заданных ключей не лежат между K_{a_i} и K_{b_i}; это, в свою очередь, есть вероятность того, что два числа, случайно выбранные из $\{1, 2, \ldots, c_i+2\}$, окажутся последовательными, а именно

$$(c_i + 1) \Big/ \binom{c_i + 2}{2}.$$

(b) Первые $n-1$ значений c_i равны нулю; затем идут $(n-2)$ единиц, $(n-3)$ двоек и т. д.; среднее значение, следовательно, равно $2\sum_{k=1}^{n}(n-k)/(k+1) = 2\sum_{k=1}^{n}((n+1)/(k+1)-1) = 2(n+1)(H_{n+1} - 1) - 2n$.

(c) Раздвоенность, характерная для слияния, приводит к тому, что алгоритм ограниченной однородной сортировки совпадает с алгоритмом однородной сортировки для этой последовательности. Пары, содержащие вершину N, имеют значения c, равные соответственно $0, 1, \ldots, N-2$; поэтому среднее число сравнений точно такое же, как и при быстрой сортировке.

66. Нет. Когда $N = 5$, никакая последовательность пар, оканчивающаяся на $(1, 5)(1, 2)(2, 3)(3, 4)(4, 5)$, не будет требовать 10 сравнений. [Интересная проблема для исследования: найти для любого конечного N однородный метод сортировки, наилучший из возможных в наихудшем случае.]

67. Гил Калаи (Gil Kalai) неофициально сообщил о найденном доказательстве минимальности для ограниченного случая. Он использовал метод, изложенный в его же статье в *Graphs and Combinatorics* **1** (1985), 65–79; однако само доказательство не опубликовано.

68. За каждый проход элемент может потерять не более одной инверсии, поэтому минимальное число проходов не может быть меньше максимального числа инверсий любого элемента исходной перестановки. При стратегии метода пузырька эта граница достигается, так как каждый проход уменьшает на единицу число инверсий каждого элемента, обладающего инверсиями (см. упр. 5.2.2–1). Мог бы потребоваться дополнительный проход, чтобы

определить, закончена ли сортировка, но формулировка этого упражнения позволяет нам не учитывать соображения такого рода.

Возможно, наше несчастье в том, что первая теорема в изучении вычислительной сложности автоматов установила "оптимальность" метода сортировки, который так плох с точки зрения его программной реализации! Положение аналогично истории с датчиками случайных чисел, когда было сделано несколько шагов назад, как только датчики, "оптимальные" с одной частной точки зрения, были рекомендованы для общего использования. (См. комментарии к выражению 3.3.3-(39).) Отсюда мораль: рассуждения об оптимальности зачастую сильно зависят от принятого уровня абстракции модели; всякие результаты, даже очень интересные, требуют осмотрительности при их практическом применении.

[Демут далее рассмотрел более общую машину с r регистрами (это ускоряет работу в r раз) и устройство, аналогичное машине Тьюринга, в котором направление движения может по желанию переключаться. Он обнаружил, что этот вид машин способен выполнять простые вставки и шейкер-сортировку; но любая такая машина с одним регистром должна в среднем совершить не менее $\frac{1}{4}(N^2 - N)$ шагов, так как каждый шаг уменьшает общее число инверсий не более чем на единицу. Наконец, он рассмотрел r-регистровые машины с произвольным доступом и вопрос сортировки с минимальным числом сравнений. Эта часть его тезисов была опубликована в *IEEE Transactions* **C-34** (1985), 296–310.]

РАЗДЕЛ 5.4

1. Можно обойтись без фазы внутренней сортировки, но при этом работа значительно замедлится, так как возрастет число считываний каждой части данных из внешней памяти и записи в нее.

2. Серии распределяются как в (1), затем на ленту 3 записываются $R_1 \ldots R_{2000000}$; $R_{2000001} \ldots R_{4000000}$; $R_{4000001} \ldots R_{5000000}$. После перемотки всех лент в результате выполнения "однопутевого" слияния на ленты T_1 и T_2 будет помещено соответственно содержимое T_3 и T_4 в (2). Затем T_1 и T_2 сливаются на T_3, информация копируется и сливается еще раз; в итоге имеем пять проходов. В общем случае эта процедура аналогична четырехленточному сбалансированному слиянию, но выполняется копирование между любыми двумя слияниями, что приводит, таким образом, к удвоенному числу проходов минус один.

3. (a) $\lceil \log_P S \rceil$. (b) $\log_B S$, где $B = \sqrt{P(T-P)}$ — так называемая "эффективная мощность слияния". При $T = 2P$ эффективная мощность равна P, при $T = 2P - 1$ она равна $\sqrt{P(P-1)} = P - \frac{1}{2} - \frac{1}{8}P^{-1} + O(P^{-2})$, что несколько меньше, чем $\frac{1}{2}T$.

4. $\frac{1}{2}T$. Если T нечетно, а P должно быть целым, то как $\lceil T/2 \rceil$, так и $\lfloor T/2 \rfloor$ дают одинаковое максимальное значение. В соответствии с упр. 3 лучше иметь $P \geq T - P$, поэтому для сбалансированного слияния мы выбираем $P = \lceil T/2 \rceil$.

РАЗДЕЛ 5.4.1

1. $087\ 154\ 170\ 426 \begin{cases} 503 \begin{cases} 503 & \infty \\ 908 & \infty \end{cases} \\ 426 \begin{cases} 426\ 653 & \infty \\ 612 & \infty \end{cases} \end{cases}$

2. Путь $\boxed{061}$—(512)—(087)—(154)—(061) был заменен путем $\boxed{612}$—(612)—(512)—(154)—(087). (По существу, мы выполняем сортировку методом пузырька от нижней части к вершине дерева вдоль этого пути.)

3. `and fourscore our seven years/ ago brought fathers forth on this/`
`a conceived continent in liberty nation new the to/ and dedicated men`
`proposition that/ all are created equal.`

4. (Эта задача несколько двусмысленна; здесь мы не очищаем внутреннюю память, пока дополнительный буфер не будет близок к переполнению.)

```
and fourscore on our seven this years/ ago brought continent fathers
forth in liberty nation new to/ a and conceived dedicated men
proposition that the/ all are created equal.
```

5. Неверно: полное двоичное дерево с P узлами определено для всех $P \geq 1$.

6. Вставить "если T = LOC($X[0]$), то переход к шагу R2; в противном случае" в начало шага R6 и исключить аналогичную фразу из шага R7.

7. Алгоритм ничего не выдает, RMAX остается равным 0.

8. Если бы любой из первых P реальных ключей оказался равным ∞, то запись пропала бы. Чтобы избежать этого, мы можем ввести переключатель, установив его так, чтобы на шаге R4 сравнение с LASTKEY первоначально не выполнялось. Затем, когда впервые окажется RQ $\neq 0$ на шаге R3, переключатель изменяет свое состояние таким образом, что далее не выполняются шаг R1 и анализ RQ на шаге R3.

9. Предположим, например, что текущая серия — восходящая, тогда как следующая должна быть нисходящей. В этом случае алгоритм R будет работать правильно при одном изменении: на шаге R6, если RN(T) = RQ > RC, следует изменить на противоположный знак сравнения KEY(LOSER(T)) с KEY(Q).

Когда RC меняется, проверки ключей на шагах R4 и R6 должны изменяться соответствующим образом.

10. Пусть $\cdot j \equiv$ LOC($X[j]$). Алгоритм R обеспечивает выполнение следующих условий при достижении шага R3, если заранее установлено LOSER($\cdot 0$) $\leftarrow$ Q и RN($\cdot 0$) $\leftarrow$ RQ. Значения LOSER($\cdot 0$), ..., LOSER($\cdot(P-1)$) представляют собой перестановку множества $\{\cdot 0, \cdot 1, \ldots, \cdot(P-1)\}$; существует перестановка указателей $\{$LOSER($\cdot j$) | RN($\cdot j$) $= 0\}$, которая соответствует текущей турнирной сетке. Другими словами, если RN($\cdot j$) равно нулю, величина KEY(LOSER($\cdot j$)) не имеет значения; можно переставить "проигравших" друг с другом. После P шагов все RN($\cdot j$) будут отличны от нуля. Таким образом, дерево будет сформировано полностью. (Ответ на указание — "да".)

Любители строгости формулировок могут возразить, что алгоритм сравнивает значения KEY, которые еще не инициализированы. Если вас это тоже смущает, можете избежать такой ситуации, установив на шаге R1 все KEY равными 0.

11. Верно. (Оба ключа принадлежат одной подпоследовательности из доказательства теоремы K.)

13. После завершения первой серии в памяти обычно остаются ключи, меньшие среднего, так как именно из-за своей малой величины они не попали в первую серию. Поэтому во вторую серию выводится большее количество меньших ключей.

14. Предположим, что снег внезапно прекращается, когда снегоочиститель находится в случайной точке u, $0 \leq u < 1$ (после достижения установившегося состояния). Тогда предпоследняя серия содержит $(1+2u-u^2)P$ записей, а последняя — u^2P. Интегрирование по du дает среднее количество записей $(2-\frac{1}{3})P$ в предпоследней серии и $\frac{1}{3}P$ — в последней.

15. Неверно. Последняя серия может быть сколь угодно длинной, но только в сравнительно редком случае, когда в момент исчерпания исходных данных все записи в памяти принадлежат одной серии.

16. Тогда и только тогда, когда каждый элемент имеет меньше P инверсий. (См. разделы 5.1.1 и 5.4.8.) Рассматривая таблицу инверсий, находим вероятность, которая равна 1 при $N \leq P$ и $P^{N-P}P!/N!$ при $N \geq P$. (Практически, однако, сортировка за один проход — не такое уж редкое явление, поскольку люди часто в целях предосторожности склонны сортировать файл при малейшем подозрении, что порядок в нем нарушен.)

17. Ровно $\lceil N/P \rceil$ серий. Все серии, кроме последней, имеют длину P. (Наихудший случай.)

18. При втором проходе ничего не изменится, так как можно показать, что k-я запись какой-либо серии меньше, чем, по крайней мере, $P+1-k$ записей предыдущей серии для $1 \le k \le P$. (Однако вряд ли существует простой способ охарактеризовать результат P'-путевого выбора с замещением, примененного после P-путевого выбора с замещением при $P' > P$.)

19. Так же, как при выводе (2), убеждаемся, что $h(x,t)\,dx = KL\,dt$, где на этот раз $h(x,t) = I + Kt$ для всех x и $P = LI$. Это влечет за собой $x(t) = L\ln((I+Kt)/I)$, так что, когда $x(T) = L$, имеем $KT = (e-1)I$. Количество снега, упавшего с момента $t=0$, составляет, следовательно, $(e-1)LI = (e-1)P$.

20. Как и в упр. 19, имеем $(I+Kt)\,dx = K(L-x)\,dt$; следовательно, $x(t) = LKt/(I+Kt)$. Количество записей в резервуаре равно

$$LI = P = P' = \int_0^T x(t)K\,dt = L(KT - I\ln((I+KT)/I));$$

следовательно, $KT = \alpha I$, где $\alpha \approx 2.14619$ — корень уравнения $1+\alpha = e^{\alpha-1}$. Длина серии есть суммарное количество снега за время $0 \le t \le T$, а именно — $LKT = \alpha P$.

21. Действуем, как описывалось в тексте раздела, но после каждой серии снегоочиститель, прежде чем вновь начать работу, ожидает, пока упадут $P - P'$ снежинок. Это означает, что $h(x(t),t)$ стало теперь не KT_1, а KT, где $T_1 - T$ есть время дополнительного падения снега. Длина серии равна LKT_1, $x(t) = L(1-e^{-t/T_1})$, $P = LKT_1e^{-T/T_1}$ и $P' = \int_0^T x(t)K\,dt = P + LK(T-T_1)$. Другими словами, длина серии $e^\theta P$ получится, если $P' = (1-(1-\theta)e^\theta)P$ при $0 \le \theta \le 1$.

22. При $0 \le t \le (\kappa-1)T$ имеем $dx \cdot h = K\,dt\,(x(t+T) - x(t))$, а при $(\kappa-1)T \le t \le T$ имеем $dx \cdot h = K\,dt\,(L - x(t))$, где h, как можно видеть, постоянно равна KT в той точке, в которой находится снегоочиститель. Отсюда следует, что при $0 \le j \le k$, $0 \le u \le 1$ и $t = (\kappa - j - u)T$ будем иметь $x(t) = L(1 - e^{u-\theta}F_j(u)/F(\kappa))$. Длина серии есть LKT — количество снега, падающего между моментами, когда снегоочистители в установившемся режиме последовательно выходят из точки 0; P есть количество убранного снега в конце последнего участка каждого снегоочистителя, а именно — $KT(L - x(\kappa T)) = LKTe^{-\theta}/F(\kappa)$; можно показать, что $P' = \int_0^{\kappa T} x(t)K\,dt$ и имеет требуемый вид.

Замечания. Оказывается, что эта формула справедлива также и для $k=0$. При $k \ge 1$ число элементов каждой серии, попадающих в резервуар *дважды*, равно $P'' = \int_0^{(\kappa-1)T} x(t)K\,dt$ и можно легко показать, что (длина серии) $- P' + P'' = (e-1)P$. Это свойство отметили Фрэйзер и Вонг. Является ли случайным совпадением то, что производящая функция для $F_k(\theta)$ так похожа на функцию из упр. 5.1.3–11?

23. Пусть $P = pP'$ и $q = 1-p$. В течение первых T_1 единиц времени падающий снег берется из числа qP' элементов, оставшихся в резервуаре после того, как вначале были удалены первые pP' элементов в случайном порядке. Когда старый резервуар станет пустым, снег снова начнет падать равномерно. Мы выбираем T_1 так, чтобы $LKT_1 = qP'$. При $0 \le t \le T_1$ имеем $h(x,t) = (p + qt/T_1)g(x)$, где $g(x)$ — вес снега, помещаемого в резервуар из точки x; при $T_1 \le t \le T$ имеем $h(x,t) = g(x) + (t-T_1)K$. При $0 \le t \le T_1$ имеем, что $g(x(t))$ есть $(q(T_1-t)/T_1)g(x(t)) + (T-T_1)K$, а при $T_1 \le t \le T$ имеем $g(x(t)) = (T-t)K$. Таким образом, $h(x(t),t) = (T-T_1)K$ при $0 \le t \le T$ и $x(t) = L(1-\exp(-t/(T-T_1)))$. Общая длина серии составляет $LK(T-T_1)$; общее число элементов, повторно возвращающихся в резервуар, равно LKT_1 (см. упр. 22); общее количество снега, счищаемого после момента T, есть $P = KT(L - x(T))$.

Таким образом, в условиях, заданных для этого упражнения, получаются серии длиной $(e^s/s)P$, если размер резервуара — $(1+(s-1)e^s/s)P$. Это значительно хуже результатов упр. 22, поскольку там содержимое резервуара используется рациональнее.

(Тот факт, что $h(x(t),t)$ есть величина постоянная в таком большом числе задач, не удивителен, ведь он эквивалентен утверждению, что элементы каждой серии, получаемой в установившемся режиме системы, распределены равномерно.)

24. (a) Работает, по существу, то же доказательство; в каждой последовательности имеются серии того же направления, что и выводные серии. (b) Вероятность, о которой говорится в указании, есть вероятность того, что серия имеет длину $n+1$ и за ней следует y; она равна $(1-x)^n/n!$, если $x>y$, и $(1-x)^n/n!-(y-x)^n/n!$, если $x\le y$. (c) Доказывается по индукции. Например, если n-я серия — восходящая, то $(n-1)$-я была нисходящей с вероятностью p, так что применяется первый интеграл. (d) Находим, что $f'(x)=f(x)-c-pf(1-x)-qf(x)$; тогда $f''(x)=-2pc$, что, в конце концов, приводит к $f(x)=c(1-qx-px^2)$, $c=6/(3+p)$. (e) Если $p>eq$, то pe^x+qe^{1-x} монотонно возрастает при $0\le x\le 1$ и $\int_0^1|pe^x+qe^{1-x}-e^{1/2}|\,dx=(p-q)(e^{1/2}-1)^2<0.43$. Если $q\le p<eq$, то pe^x+qe^{1-x} лежит между $2\sqrt{pqe}$ и $p+qe$, так что $\int_0^1|pe^x+qe^{1-x}-\frac12(p+qe+2\sqrt{pqe})|\,dx\le\frac12(\sqrt{p}-\sqrt{qe})^2<0.4$, и если $p<q$, можно использовать "симметричное" рассуждение. Таким образом, для всех p и q существует константа C, такая, что $\int_0^1|pe^x+qe^{1-x}-C|\,dx<0.43$. Пусть $\delta_n(x)=f_n(x)-f(x)$. Тогда $\delta_{n+1}(y)=(1-e^{y-1})\int_0^1(pe^x+qe^{1-x}-C)\,\delta_n(x)\,dx+p\int_0^{1-y}e^{y-1+x}\delta_n(x)\,dx+q\int_y^1 e^{y-x}\delta_n(x)\,dx$. Следовательно, если $\delta_n(y)\le\alpha_n$, $|\delta_{n+1}(y)|\le(1-e^{y-1})\cdot 1.43\alpha_n<0.91\alpha_n$. (f) При всех $n\ge 0$ величина $(1-x)^n/n!$ есть вероятность того, что длина серий превышает n. (g) $\int_0^1(pe^x+qe^{1-x})f(x)\,dx=6/(3+p)$.

26. (a) Рассмотрите число перестановок из $n+r+1$ элементов с n левосторонними минимумами, причем крайний справа элемент не является наименьшим. (b) Используйте свойство

$$\sum_{1\le k<n}\left[{k \atop k-r}\right]k=\left[{n \atop n-r-1}\right]$$

по определению чисел Стирлинга в приложении Б. (c) Добавьте к средней длине $r+1$, используя для получения равенства $\sum_{n\ge 0}\left[{n+r \atop n}\right](n+r)/(n+r+1)!=1$ тот факт, что $\sum_{n\ge 0}\left[{n+r \atop n}\right]/(n+r-1)!$.

Формула в (b) получена в работе P. Appell, *Archiv der Math. und Physik* **65** (1880), 171–175. Соответственно имеем $\left[\left[{r \atop k}\right]\right]=(r+k)!\,[x^kz^r]\,e^{xf(z)}$, где $f(z)=z/2+z^2/3+\cdots=-z^{-1}\ln(1-z)-1$; следовательно, $c_r=[z^r]\,(r+1+f(z))\,e^{f(z)}$. Число неупорядоченностей n объектов, которые имеют k циклов, иногда обозначаемое как $\left[{n \atop k}\right]_{\ge 2}$, равно $\left[\left[{n-k \atop k}\right]\right]$; см. J. Riordan, *An Introduction to Combinatorial Analysis* (Wiley, 1958), §4.4. (Имеется русский перевод: Риордан Дж. *Введение в комбинаторный анализ.* — М.: Изд-во иностр. лит., 1963.)

27. Если при $0\le\theta\le 1$ $P'/P=2(e^{-\theta}-1+\theta)/(1-2\theta+\theta^2+2\theta e^{-\theta})$, то средняя длина серии в установившемся режиме будет равна $2P/(1-2\theta+\theta^2+2\theta e^{-\theta})$. [См. *Information Processing Letters* **21** (1985), 239–243.]

Добосевич обратил также внимание на то, что можно продолжать использование механизма выбора с замещением, поскольку можно вводить данные из начала очереди во вспомогательном буфере и одновременно выводить их с конца этой очереди. Например, если $P'=.5P$ и продолжается выбор с замещением до тех пор, пока текущая серия не будет содержать $.209P$ записей, средняя длина серии при такой модификации возрастает от примерно $2.55P$ до примерно $2.61P$. Если же $P'=P$ и продолжается выполнение выбора с замещением до тех пор, пока в текущей серии не останется только $.314P$ записей, то средняя длина серии возрастает от eP до примерно $3.034P$. [См. *Comp. J.* **27**

(1984), 334–339, где представлен даже более эффективный метод, названный "слиянием с замещением".]

28. Для многопутевого слияния эта проблема относительна проста, поскольку P остается постоянным и записи обрабатываются последовательно в каждом файле. Однако при формировании начальных серий предпочтительнее изменять число записей в памяти в зависимости от их длин. Можно было бы применить пирамиду из такого числа записей, которые помещаются в памяти, используя динамическое распределение памяти, как описано в разделе 2.5. В работе M. A. Goetz, *Proc. AFIPS Joint Computer Conf.* **25** (1964), 602–604, предложен другой подход: разбить каждую запись на части фиксированного размера, которые связаны между собой (они располагаются в листьях деревьев, и только "головная" часть участвует в турнире).

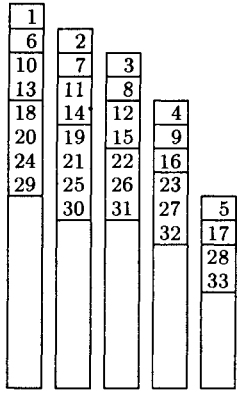
29. Верхние 2^k узлов проигравших переходят на соответствующие позиции основных узлов. Оставшиеся узлы проигравших образуют 2^k поддеревьев с $2^n - 1$ узлами в каждом; они подключаются к основным узлам в симметричном порядке: крайнее слева *до*дерево — к крайнему слева основному узлу и т. д. [См. K. Efe, N. Eleser, *Acta Informatica* **34** (1997), 429–447.]

30. Предположим, что к t основным узлам *подключен* 2^n-узловой подграф полного 2^{n+k}-узлового дерева проигравших. В этом дереве имеется один узел на уровне 0 и 2^{l-1} узлов на уровне l при $1 \le l \le n+k$. Поддерево с корнем на уровне $l \ge 1$ имеет $2^{n+k+1-l} - 1$ узлов. Таким образом, все корни t несвязанных 2^n-узловых деревьев должны находиться на уровнях $\le k$. Каждое из этих поддеревьев должно содержать минимум один узел на уровне k, поскольку существует только $2^{k-1} < 2^n$ узлов на уровнях $< k$. Отсюда следует, что $t \le 2^{k-1}$. Но количество ребер основного графа равно, по меньшей мере, $t + 2(2^k - t) - 1$, как следует из (ii) и (iii), поскольку существует не меньше этого количества узлов проигравших, родитель которых имеет отличную картину в основном графе.

[Необходимо предполагать, что $n \ge k$: если $n = k - 1$, то существует соответствующий основной граф с $2^k + 2^{k-1} - 2$ ребрами.]

РАЗДЕЛ 5.4.2

1.



2. После первой фазы слияния все оставшиеся фиктивные серии будут находиться на ленте T и их будет самое большее $a_n - a_{n-1} \le a_{n-1}$. Следовательно, все они исчезнут в течение второй фазы слияния.

3. Имеем $(\mathtt{D[1]}, \mathtt{D[2]}, \ldots, \mathtt{D}[T]) = (a_n - a_{n-P}, a_n - a_{n-P+1}, \ldots, a_n - a_n)$, так что выполнение интересующего нас условия следует из того, что последовательность a неубывающая. Это

условие важно для правильной работы алгоритма, так как на шагах D2 и D3 никогда D[$j+1$] не уменьшается чаще, чем D[j].

4. $(1-z-\cdots-z^5)a(z)=1$ в силу (3). Далее, $t(z)=\sum_{n\ge1}(a_n+b_n+c_n+d_n+e_n)z^n=(z+\cdots+z^5)a(z)+(z+\cdots+z^4)a(z)+\cdots+za(z)=(5z+4z^2+3z^3+2z^4+z^5)a(z)$.

5. Пусть $g_p(z)=(z-1)f_p(z)=z^{p+1}-2z^p+1$ и $h_p(z)=z^{p+1}-2z^p$. Теорема Руше (Rouché) [*J. École Polytechnique* **21**, 37 (1858), 1–34] утверждает: $h_p(z)$ и $g_p(z)$ имеют равное число корней внутри окружности $|z|=1+\epsilon$ при условии, что $|h_p(z)|>|h_p(z)-g_p(z)|=1$ на окружности. Если $\phi^{-1}>\epsilon>0$ имеем $|h_p(z)|\ge(1+\epsilon)^p(1-\epsilon)>(1+\phi^{-1})^2(1-\phi^{-1})=1$. Следовательно, g_p имеет p корней с абсолютной величиной ≤1. Они различны, так как $\gcd(g_p(z),g_p'(z))=\gcd(g_p(z),(p+1)z-2p)=1$. [*AMM* **67** (1960), 745–752.]

6. Положим $c_0=-\alpha p(\alpha^{-1})/q'(\alpha^{-1})$. Тогда $p(z)/q(z)-c_0/(1-\alpha z)$ аналитична в круге $|z|\le R$ для некоторого $R>|\alpha|^{-1}$; следовательно, коэффициенты $[z^n]p(z)/q(z)=c_0\alpha^n+O(R^{-n})$. Значит, $\ln S=n\ln\alpha+\ln c_0+O((\alpha R)^{-n})$; а из $n=(\ln S/\ln\alpha)+O(1)$ следует, что $O((\alpha R)^{-n})=O(S^{-\epsilon})$. Аналогично положим $c_1=\alpha^2p(\alpha^{-1})/q'(\alpha^{-1})^2$, $c_2=-\alpha p'(\alpha^{-1})/q'(\alpha^{-1})^2+\alpha p(\alpha^{-1}q''(\alpha^{-1})/q'(\alpha^{-1})^3$ и рассмотрим $p(z)/q(z)^2-c_1/(1-\alpha z)^2-c_2/(1-\alpha z)$.

7. Пусть $\alpha_p=2x$ и $z=-1/2^{p+1}$. Тогда $x^{p+1}=x^p+z$ и в результате получается сходящийся ряд $\alpha_p=2\sum_{k\ge0}\binom{1-kp}{k}z^k/(1-kp)=2-2^{-p}-p2^{-2p-1}+O(p^22^{-3p})$, как следует из формулы 1.2.6–(25).

Замечание. Отсюда следует, что величина ρ в упр. 6 становится примерно равной $\log_4 S$ с ростом p. Аналогично для табл. 5 и 6 коэффициент c приближается к $1/((\phi+2)\ln\phi)$ при большом числе лент.

8. Очевидно, $N_0^{(p)}=1$, $N_m^{(p)}=0$ для $m<0$. Рассматривая все варианты для первого слагаемого, получаем $N_m^{(p)}=N_{m-1}^{(p)}+\cdots+N_{m-p}^{(p)}$ при $m>0$. Следовательно, $N_m^{(p)}=F_{m+p-1}^{(p)}$. [*Lehrbuch der Combinatorik* (Leipzig: Teubner, 1901), 136–137.]

9. Рассмотрим положение крайнего слева нуля, если таковой имеется; находим, что $K_m^{(p)}=F_{m+p}^{(p)}$. *Замечание.* Существует простое взаимно однозначное соответствие между такими последовательностями нулей и единиц и изображениями $m+1$, рассмотренными в упр. 8: добавьте 0 к правому концу последовательности и посмотрите на положение всех нулей.

10. *Лемма.* Если $n=F_{j_1}^{(p)}+\cdots+F_{j_m}^{(p)}$ является таким представлением, где $j_1>\cdots>j_m\ge p$, то $n<F_{j_1+1}^{(p)}$. *Доказательство.* Этот результат очевиден, если $m<p$. В противном случае пусть k есть минимальное число, такое, что $j_k>j_{k+1}+1$; имеем $k<p$ и по индукции $F_{j_{k+1}}^{(p)}+\cdots+F_{j_m}^{(p)}<F_{j_k-1}^{(p)}$; следовательно, $n<F_{j_1}^{(p)}+\cdots+F_{j_1-k-1}^{(p)}\le F_{j_1+1}^{(p)}$.

Искомое теперь может быть доказано индукцией по n. Если $n>0$, то пусть j будет максимальным числом, таким, что $F_j^{(p)}\le n$. Лемма показывает, что любое представление n должно состоять из $F_j^{(p)}$ плюс представление $n-F_j^{(p)}$. По индукции $n-F_j^{(p)}$ имеет единственное представление нужного вида, и это представление не содержит всех чисел $F_{j-1}^{(p)},\ldots,F_{j-p+1}^{(p)}$, так как j максимально.

Замечания. Случай, когда $p=2$, упомянутый в работе E. Zeckendorf, *Simon Stevin* **29** (1952), 190–195, был рассмотрен в упр. 1.2.8–34. Имеется простой алгоритм перехода от представления n к представлению $n+1$, работающий с последовательностью нулей и единиц $c_j\ldots c_1c_0$, такой, что $n=\sum c_jF_{j+p}^{(p)}$. Например, если $p=3$, мы смотрим на правые цифры и заменяем $\ldots0$ на $\ldots1$, $\ldots01$ на $\ldots10$, $\ldots011$ на $\ldots100$; затем осуществляем "перенос" влево, если это необходимо, заменяя $\ldots0111\ldots$ на $\ldots1000\ldots$. (См. последовательности нулей и единиц в упр. 9 в том порядке, в котором они записаны.) Подобная система счисления была исследована в работе W. C. Lynch, *Fibonacci Quarterly* **8** (1970), 6–22. Автор приводит

очень интересный способ ее применения для управления и фазой распределения, и фазой слияния многофазной сортировки.

12. k-я степень содержит точные распределения для уровней с $k-4$ до k-го в последовательных строках с наибольшими элементами справа.

13. Доказывается индукцией по уровню.

14. (a) $n(1) = 1$, поэтому будем считать, что $k > 1$. Закон $T_{nk} = T_{(n-1)(k-1)} + \cdots + T_{(n-P)(k-1)}$ показывает, что $T_{nk} \le T_{(n+1)k}$ тогда и только тогда, когда $T_{(n-P)(k-1)} \le T_{n(k-1)}$. Пусть r — произвольное положительное целое и пусть n' — минимальное число, такое, что $T_{(n'-r)(k-1)} > T_{n'(k-1)}$; тогда $T_{(n-r)(k-1)} \ge T_{n(k-1)}$ для всех $n \ge n'$, поскольку это тривиально для $n \ge n(k-1)+r$, а иначе $T_{(n-r)(k-1)} \ge T_{(n'-r)(k-1)} \ge T_{n'(k-1)} \ge T_{n(k-1)}$. (b) То же рассуждение при $r = n - n'$ показывает, что $T_{n'k'} < T_{nk'}$ влечет за собой $T_{(n'-j)k'} \le T_{(n-j)k'}$ для всех $j \ge 0$. Значит, из рекуррентного соотношения следует, что $T_{(n'-j)k} \le T_{(n-j)k}$ для всех $j \ge 0$ и $k \ge k'$. (c) Пусть $\ell(S)$ — наименьшее n, такое, что $\Sigma_n(S)$ принимает свое минимальное значение. Требуемая последовательность M_n существует тогда и только тогда, когда $\ell(S) \le \ell(S+1)$ для всех S. Предположим, что $n = \ell(S) > \ell(S+1) = n'$, так что $\Sigma_n(S) < \Sigma_{n'}(S)$ и $\Sigma_n(S+1) \ge \Sigma_{n'}(S+1)$. Существует наименьшее S', такое, что $\Sigma_n(S') < \Sigma_{n'}(S')$; имеем $m = \Sigma_n(S') - \Sigma_n(S'-1) < \Sigma_{n'}(S') - \Sigma_{n'}(S'-1) = m'$. Тогда $\sum_{k=1}^{m} T_{n'k} < S' \le \sum_{k=1}^{m} T_{nk}$; следовательно, существует некоторое $k' \le m$, такое, что $T_{n'k'} < T_{nk'}$. Аналогично имеем $l = \Sigma_n(S+1) - \Sigma_n(S) > \Sigma_{n'}(S+1) - \Sigma_{n'}(S) = l'$; значит, $\sum_{k=1}^{l'} T_{n'k} \ge S+1 > \sum_{k=1}^{l'} T_{nk}$. Поскольку $l' \ge m' > m$, существует некоторое $k > m$, такое, что $T_{n'k} > T_{nk}$. Но это противоречит п. (b).

15. Эта теорема была доказана Д. Э. Зэйвом (D. A. Zave), статья которого упоминается в тексте раздела.

16. Д. Э. Зэйв показал, что число вводимых (и выводимых) записей равно $S\log_{T-1} S + \frac{1}{2} S \log_{T-1}\log_{T-1} S + O(S)$.

17. Пусть $T = 3$; $A_{11}(x) = 6x^6 + 35x^7 + 56x^8 + \cdots$, $B_{11}(x) = x^6 + 15x^7 + 35x^8 + \cdots$, $T_{11}(x) = 7x^6 + 50x^7 + 91x^8 + 64x^9 + 19x^{10} + 2x^{11}$. Оптимальное распределение для $S = 144$ требует 55 серий на T2. Это обязательно приводит к неоптимальному распределению для $S = 145$. Д. Э. Зэйв изучил процедуры такого вида, близкие к оптимальным.

18. Пусть $S = 9$, $T = 3$. Рассмотрим следующие две схемы.

Оптимальное многофазное слияние				Альтернатива			
T1	T2	T3	Стоимость	T1	T2	T3	Стоимость
$0^2 1^6$	$0^2 1^3$	—		$0^1 1^6$	$0^1 1^3$	—	
1^3	—	$0^2 2^3$	6	1^3	—	$0^1 2^3$	6
—	$1^2 3^1$	2^2	5	—	$1^1 3^2$	2^1	7
3^2	3^1	—	6	3^1	3^2	—	3
3^1	—	6^1	6	—	3^1	6^1	6
—	9^1	—	9	9^1	—	—	9
			32				31

(Еще один способ улучшения "оптимального" многофазного метода состоит в пересмотре того, в каких местах выводной ленты появляются фиктивные серии на каждой фазе слияния. Например, результат слияния $0^2 1^3$ с $0^2 1^3$ можно было бы рассматривать как $2^1 0^1 2^1 0^1 2^1$ вместо $0^2 2^3$. Таким образом, остается много нерешенных вопросов, связанных с оптимальностью.)

19.

Уровень	T1	T2	T3	T4	Сумма	Окончательный результат на
0	1	0	0	0	1	T1
1	0	1	1	1	3	T6
2	1	1	1	0	3	T5
3	1	2	1	1	5	T4
4	2	2	2	1	7	T3
5	2	4	3	2	11	T2
6	4	5	4	2	15	T1
7	5	8	6	4	23	T6
. . .	. . .	. . .	. . .	. . .	. . .	. . .
n	a_n	b_n	c_n	d_n	t_n	$\mathrm{T}(k)$
$n+1$	b_n	c_n+a_n	d_n+a_n	a_n	t_n+2a_n	$\mathrm{T}(k-1)$

20. $a(z)=1/(1-z^2-z^3-z^4)$, $t(z)=(3z+3z^2+2z^3+z^4)/(1-z^2-z^3-z^4)$, $\sum_{n\ge1}T_n(x)z^n = x(3z+3z^2+2z^3+z^4)/(1-x(z^2+z^3+z^4))$. $D_n=A_{n-1}+1$, $C_n=A_{n-1}A_{n-2}+1$, $B_n = A_{n-1}A_{n-2}A_{n-3}+1$, $A_n=A_{n-2}A_{n-3}A_{n-4}+1$.

21. 333343333332322 3333433333323 33334333333 3333433 333323 T5

22. $t_n-t_{n-1}-t_{n-2}=-1+3[n \bmod 3=1]$. (Это соотношение, подобное соотношению Фибоначчи, следует из того, что $1-z^2-2z^3-z^4=(1-\phi z)(1-\hat{\phi}z)(1-\omega z)(1-\overline{\omega}z)$, где $\omega^3=1$.)

23. Вместо (25) длины серий в течение первой половины n-й фазы слияния будут s_n, а в течение второй половины — t_n, где $s_n=t_{n-2}+t_{n-3}+s_{n-3}+s_{n-4}$, $t_n=t_{n-2}+s_{n-2}+s_{n-3}+s_{n-4}$. Здесь мы полагаем $s_n=t_n=1$ при $n\le0$. [В общем случае, если v_{n+1} является суммой первых $2r$ членов из $u_{n-1}+\cdots+v_{n-P}$, имеем $s_n=t_n=t_{n-2}+\cdots+t_{n-r}+2t_{n-r-1}+t_{n-r-2}+\cdots+t_{n-P}$. Если v_{n+1} есть сумма первых $2r-1$ членов, то $s_n=t_{n-2}+\cdots+t_{n-r-1}+s_{n-r-1}+\cdots+s_{n-P}$, $t_n=t_{n-2}+\cdots+t_{n-r}+s_{n-r}+\cdots+s_{n-P}$.]

Вместо (27) и (28)

$$\begin{aligned}
A_n &= (U_{n-1}V_{n-1}U_{n-2}V_{n-2}U_{n-3}V_{n-3}U_{n-4}V_{n-4})+1,\ \ldots,\\
D_n &= (U_{n-1}V_{n-1})+1,\\
E_n &= (U_{n-2}V_{n-2}U_{n-3})+1,\\
V_{n+1} &= (U_{n-1}V_{n-1}U_{n-2})+1,\\
U_n &= (V_{n-2}U_{n-3}V_{n-3}U_{n-4}V_{n-4})+1.
\end{aligned}$$

25.

1^{16}	1^8	—	1^8
1^{12}	1^4	R	$1^8 2^4$
1^8	—	2^4	R
. . .	. . .	. . .	. . .
R	$8^1 16^1$	8^1	8^0
16^0	R	8^1	—
16^1	16^1	8^0	R
R	16^1	—	24^0
16^1	16^1	R	$24^0 32^0$
16^0	16^0	32^1	(R)

26. Если сортируются 2^n начальных серий, то во время слияния обрабатываются $n\cdot2^n$-е серии. Каждая половина фазы (за немногими исключениями) сливает 2^{n-2} и перематывает 2^{n-1}. Если сортируются 2^n+2^{n-1} начальных серий, то обрабатываются во время слияния

$n \cdot 2^n + (n-1) \cdot 2^{n-1}$. Каждая половина фазы (за немногими исключениями) сливает 2^{n-2} или 2^{n-1} и перематывает $2^{n-1} + 2^{n-2}$ начальных серий.

27. Стратегия работает тогда и только тогда, когда наибольший общий делитель чисел распределения равен 1. Например, пусть имеется шесть лент; если мы распределим (a, b, c, d, e) на ленты T1–T5, где $a \geq b \geq c \geq d \geq e > 0$, то после первой фазы получится распределение $(a-e, b-e, c-e, d-e, e)$ и $\gcd(a-e, b-e, c-e, d-e, e) = \gcd(a, b, c, d, e)$. (Любой общий делитель одного из этих множеств чисел делит также числа другого множества.) Рассматриваемый процесс уменьшает количество серий на каждой фазе до тех пор, пока на одной ленте не останется $\gcd(a, b, c, d, e)$ серий.

[Заметим, что эти не многофазные распределения иногда оказываются лучше многофазного при определенном расположении фиктивных серий, как показано в упр. 18. Это свойство впервые было замечено Б. Сэкманом примерно в 1963 году.]

28. Мы получаем любую такую пятерку (a, b, c, d, e), начав с $(1, 0, 0, 0, 0)$ и выполнив ровно n раз следующую операцию: выбрать x из $\{a, b, c, d, e\}$ и добавить x к каждому из остальных четырех элементов (a, b, c, d, e).

Чтобы доказать, что $a + b + c + d + e \leq t_n$, докажем по индукции, что если $a \geq b \geq c \geq d \geq e$, то всегда имеем $a \leq a_n$, $b \leq b_n$, $c \leq c_n$, $d \leq d_n$, $e \leq e_n$. В предположении, что сказанное справедливо для уровня n, это также должно быть справедливо для уровня $n+1$, поскольку распределения $(n+1)$-го уровня суть $(b+a, c+a, d+a, e+a, a)$, $(a+b, c+b, d+b, e+b, b)$, $(a+c, b+c, d+c, e+c, c)$, $(a+d, b+d, c+d, e+d, d)$, $(a+e, b+e, c+e, d+e, e)$.

30. Дж. А. Мортенсон (J. A. Mortenson) свел результаты вычислений в следующую таблицу.

Уровень	$T=5$	$T=6$	$T=7$	$T=8$	$T=9$	$T=10$	
1	2	2	2	2	2	2	M_1
2	4	5	6	7	8	9	M_2
3	4	5	6	7	8	9	M_3
4	8	8	10	12	14	16	M_4
5	10	14	18	17	20	23	M_5
6	18	20	26	27	32	31	M_6
7	26	32	46	47	56	42	M_7
8	44	53	74	82	92	92	M_8
9	68	83	122	111	138	139	M_9
10	112	134	206	140	177	196	M_{10}
11	178	197	317	324	208	241	M_{11}
12	290	350	401	488	595	288	M_{12}
13	466	566	933	640	838	860	M_{13}
14	756	917	1371	769	1064	1177	M_{14}
15	1220	1481	1762	2078	1258	1520	M_{15}
16	1976	2313	4060	2907	3839	1821	M_{16}

31. [*Random Structures & Algorithms*, **5** (1994), 102–104.] $K_d(n) = F_{n-2}^{(d)} = N_{n-d-1}^{(d)}$. Имеем $n - d - 1 = a_1 + \cdots + a_r$, если в дереве $r+1$ листьев и $(k+1)$-й лист имеет $a_k - 1$ предков, отличных от предков первых k листьев. (Приведенные семь примеров деревьев соответствуют суммам в правой части 1+1+1+1, 1+1+2, 1+2+1, 1+3, 2+1+1, 2+2 и 3+1.)

РАЗДЕЛ 5.4.3

1. Выясняя с помощью табл. 5.4.2–6, сколько раз в среднем обрабатывается каждая запись, видим, что многофазный метод с расщеплением лент лучше, если имеется 6, 7 или 8 лент.

2. Методы, по существу, тождественны, если число начальных серий является числом Фибоначчи; в противном случае способ распределения фиктивных серий лучше у многофазного метода. Каскадный алгоритм помещает 1 на T1, 1 на T2, 1 на T1, 2 на T2, 3 на T1, 5 на T2 и т. д. Поэтому на шаге C8 никогда не обнаружится, что $D[p-1] = M[p-1]$, если $p = 2$. В результате все фиктивные серии оказываются на одной ленте, а это менее эффективно, чем метод алгоритма 5.4.2D.

3. (Распределение заканчивается после того, как 12 серий помещаются на T3 на шаге (3, 3).)

T1	T2	T3	T4	T5	T6
1^{26}	1^{21}	1^{24}	1^{14}	1^{15}	—
1^{5}	—	1^{12}	$1^{2}2^{7}$	1^{15}	$2^{2}4^{12}$
8^{4}	$6^{2}9^{3}$	5^{2}	6^{3}	1^{1}	—
—	9^{1}	23^{1}	17^{1}	25^{1}	26^{1}
100^{1}	—	—	—	—	—

4. Доказывается по индукции (см. упр. 5.4.2–28).

5. Если имеется a_n начальных серий, то k-й проход выводит a_{n-k} серий длиной a_k, затем b_{n-k} — длиной b_k и т. д.

6.

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

7. $e_2e_{n-2} + e_3e_{n-3} + \cdots + e_ne_0$ длин начальных серий (см. упр. 5), что может быть также записано в виде $a_1a_{n-3} + a_2a_{n-4} + \cdots + a_{n-2}a_0$; это коэффициент при $[z^{n-2}]$ в $(A(z)^2 - A(z))$.

8. Знаменатель $A(z)$ имеет различные корни, и его степень выше степени числителя, поэтому $A(z) = \sum q_3(\rho)/(1-\rho z)\rho(1-q_4'(\rho))$, где сумма берется по всем корням ρ уравнения $q_4(\rho) = \rho$. Этот специальный вид ρ полезен при вычислении $q_3(\rho)$ и $q_4'(\rho)$.

9. Эти формулы справедливы для всех *больших* n из (8) и (12), если иметь в виду значение $q_m(2\sin\theta_k)$. Чтобы показать, что они справедливы при всех n, необходимо знать, что $q_{m-1}(z)$ есть частное от деления $q_{r-1}(z)q_m(z)$ на $q_r(z) - z$ при $0 \le m < r$. Это можно доказать, либо используя (10) и учитывая, что при сокращении степень $q_{r-1}(z)q_m(z) - q_r(z)q_{m-1}(z)$ снижается, либо учитывая, что из результата упр. 5 следует, что $A(z)^2 + B(z)^2 + \cdots + E(z)^2 \to 0$ при $z \to \infty$, либо найдя явную формулу для числителей $B(z)$, $C(z)$ и т. д.

10. $E(z) = r_1(z)A(z)$; $D(z) = r_2(z)A(z) - r_1(z)$; $C(z) = r_3(z)A(z) - r_2(z)$; $B(z) = r_4(z)A(z) - r_3(z)$; $A(z) = r_5(z)A(z) + 1 - r_4(z)$. Таким образом, $A(z) = (1 - r_4(z))/(1 - r_5(z))$. [Обратите внимание на то, что $r_m(2\sin\theta) = \sin(2m\theta)/\cos\theta$; следовательно, $r_m(z)$ является многочленом Чебышева $(-1)^{m+1}U_{2m-1}(z/2)$.]

11. Докажите, что $f_m(z) = q_{\lfloor m/2\rfloor}(z) - r_{\lceil m/2\rceil}(z)$ и что $f_m(z)f_{m-1}(z) = 1 - r_m(z)$. Затем используйте результат упр. 10. (Это выражение для знаменателя в явном виде впервые было выведено Девидом Е. Фергюсоном (David E. Ferguson).)

13. См. упр. 5.4.6–6.

РАЗДЕЛ 5.4.4

1. *Сначала* (перед выводом восходящей серии) следует записать концевую запись, содержащую $-\infty$. (Запись с ключом $+\infty$ по-прежнему должна располагаться в конце серии, если только мы собираемся когда-либо считывать ее в прямом направлении, например на последнем проходе.) Для нисходящих серий поменять ролями $-\infty$ и $+\infty$.

2. Наименьшее число на уровне $n+1$ равно наибольшему на уровне n; следовательно, столбцы являются неубывающими независимо от того, как переставлены числа в любой отдельной строке.

3. На самом деле в процессе слияния первая серия на T2–T6 всегда будет нисходящей, а на T1 — восходящей (по индукции).

4. Такой метод требует нескольких операций "копирования" на втором и третьем проходах; дополнительные затраты приблизительно равны $(\log 2)/(\log \rho)$ проходам, где ρ — "отношение роста" (см. табл. 5.4.2–1).

5. Если α — цепочка, обозначим через α^R ее обращение.

Уровень	T1	T2	T3	T4	T5
0	0	—	—	—	—
1	1	1	1	1	1
2	12	12	12	12	2
3	1232	1232	1232	232	32
4	12323432	12323432	2323432	323432	3432
. . .	. . .	. . .	. . .	. . .	. . .
n	A_n	B_n	C_n	D_n	E_n
$n+1$	$B_n(A_n^R+1)$	$C_n(A_n^R+1)$	$D_n(A_n^R+1)$	$E_n(A_n^R+1)$	A_n^R+1

2				
3	2			
4	3	2		
3	4	3		
4	3	4	2	
5	4	3	3	
4	5	4	4	
3	4	5	3	
2	3	4	4	2
3	2	3	5	3
4	3	2	4	4
3	4	3	3	3
2	3	4	2	4
3	2	3	3	5
2	3	2	4	4
1	2	3	3	3

Имеем

$$\begin{aligned}
E_n &= A_{n-1}^R + 1,\\
D_n &= A_{n-2}^R A_{n-1}^R + 1,\\
C_n &= A_{n-3}^R A_{n-2}^R A_{n-1}^R + 1,\\
B_n &= A_{n-4}^R A_{n-3}^R A_{n-2}^R A_{n-1}^R + 1,\\
A_n &= A_{n-5}^R A_{n-4}^R A_{n-3}^R A_{n-2}^R A_{n-1}^R + 1\\
&= n - Q_n,
\end{aligned}$$

где

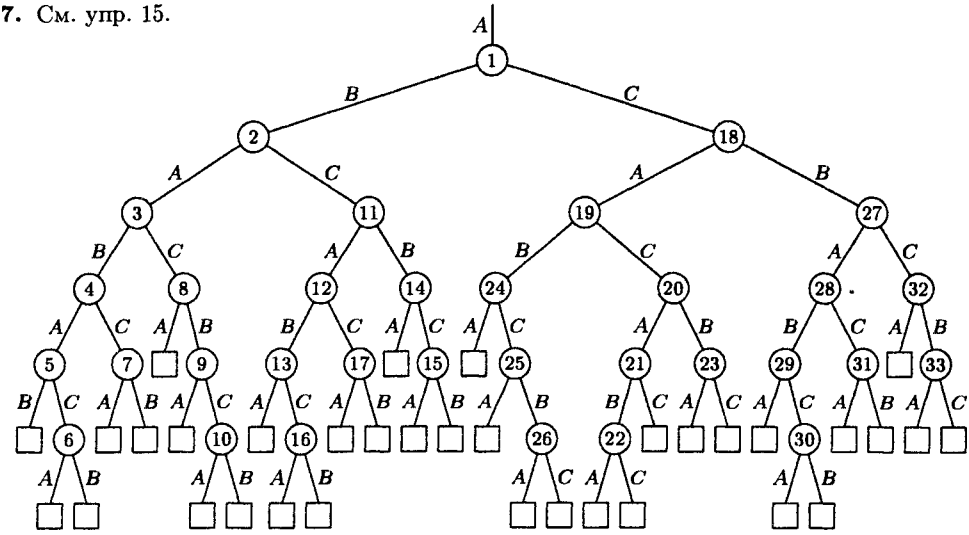
$$Q_n^R = Q_{n-1}(Q_{n-2}+1)(Q_{n-3}+2)(Q_{n-4}+3)(Q_{n-5}+4), \qquad n \geq 1,$$

$Q_0 = 0$ и цепочка $Q_n = \epsilon$ при $n < 0$.

Цепочки A_n, B_n, ... содержат те же элементы, что и соответствующие цепочки из раздела 5.4.2, но в другом порядке. Заметим, что соседние числа слияний всегда различаются на 1. Начальная серия должна иметь тип A тогда и только тогда, когда число ее слияний четно, D — если нечетно. Простые схемы распределения, такие, как в алгоритме 5.4.2D, уступают в эффективности размещению фиктивных серий в позиции с большими числами слияний; поэтому, вероятно, выгодно вычислять Q_n между фазами 1 и 2, чтобы облегчить управление процессом размещения фиктивных серий.

6. $y^{(4)} = (+1, +1, -1, +1)$
$y^{(3)} = (+1, \ \ 0, -1, \ \ 0)$
$y^{(2)} = (+1, -1, +1, +1)$
$y^{(1)} = (-1, +1, +1, +1)$
$y^{(0)} = (\ \ 1, \ \ 0, \ \ 0, \ \ 0)$

7. См. упр. 15.



Число 34 является, вероятно, наименьшим числом Фибоначчи F_n, для которого многофазный метод не дает оптимального слияния с обратным чтением для F_n начальных серий на трех лентах. Это дерево имеет длину внешнего пути 178, что на 2 лучше, чем в многофазном методе, для которого соответствующий параметр равен 176.

8. Для $T = 4$ дерево с длиной внешнего пути 13 не является T-lifo-деревом, и любое дерево с длиной внешнего пути 14 включает однопутевое слияние.

9. Используя результат упр. 2.3.4.5–6, можно рассмотреть полное $(T-1)$-арное дерево; степень "последнего" внутреннего узла лежит между 2 и $T-1$. Если в нем имеется $(T-1)^q - m$ внешних узлов, то $\lfloor m/(T-2) \rfloor$ из них находятся на уровне $q-1$, а оставшиеся — на уровне q.

11. Верно, что доказывается индукцией по числу начальных серий. Если правильное распределение S серий и две соседние серии находятся в одинаковом направлении, то существует нужное распределение меньшего, чем S, числа серий, но его не существует при $S = 1$.

12. Условия (a) и (b) очевидны. Если имеется какая-нибудь конфигурация в (4) для некоторого имени ленты A и некоторых $i < j < k$, то узел j должен быть в поддереве ниже узла i и слева от узла k по определению прямого порядка. Следовательно, случай "j — l" не может иметь места и A должно быть "специальным" именем, так как оно появляется на внешней ветви. Но это противоречит тому, что специальное имя, как мы предположили, находится на крайней слева ветви ниже узла i.

13. Узлы, пронумерованные 4, 7, 11, 13, можно преобразовать во внешние, и по отношению к ним не нужно использовать однопутевое слияние. В результате длина внешнего пути будет на единицу больше, чем в случае многофазного дерева.

15. Назовем ленты A, B и C. Построим несколько видов деревьев, каждый из которых характеризуется определенной структурой корня и листьев (внешних узлов).

Тип $r(A)$	Корень A
Тип $s(A,C)$	Корень A; нет C-листьев
Тип $t(A)$	Корень A; нет A-листьев
Тип $u(A,C)$	Корень A; нет C-листьев, нет составных B-листьев
Тип $v(A,C)$	Корень A; нет C-листьев, нет составных A-листьев
Тип $w(A,C)$	Корень A; нет A-листьев, нет составных C-листьев

"Составной лист" — это лист, "брат" которого не является листом. Можно получить 3-lifo-дерево типа $r(A)$, вырастив сначала его левое поддерево типа $s(B,C)$, а затем — правое поддерево типа $r(C)$. Аналогично тип $s(A,C)$ получается из типов $s(B,C)$ и $t(C)$; тип $u(A,C)$ — из $v(B,C)$ и $w(C,B)$; тип $v(A,C)$ — из $u(B,C)$ и $w(C,A)$. Можно вырастить 3-lifo-дерево типа $t(A)$, левое поддерево которого имеет тип $u(B,A)$, а правое поддерево — тип $s(C,A)$, позволив вырасти его левому поддереву, за исключением его (несоставных) C-листьев и его правого поддерева. В этот момент левое поддерево имеет только A- и B-листья, так что мы можем вырастить правое поддерево всего дерева, затем выбросить A-листья из левого-левого поддерева и, наконец, вырастить левое-правое поддерево. Аналогично дерево типа $w(A,C)$ можно образовать из $u(B,A)$ и $v(C,A)$. [Дерево из упр. 7 есть $r(A)$-дерево, построенное этим способом.]

Пусть $r(n),\ldots,w(n)$ обозначают минимальную длину внешнего пути среди всех деревьев соответствующего типа с n-листьями, построенных с помощью такой процедуры. Имеем $r(1)=s(1)=u(1)=0$, $r(2)=t(2)=w(2)=2$, $t(1)=v(1)=w(1)=s(2)=u(2)=v(2)=\infty$. Для $n\ge 3$ имеем

$$\begin{aligned} r(n) &= n+\min\nolimits_k(s(k)+r(n-k)), & u(n) &= n+\min\nolimits_k(v(k)+w(n-k)),\\ s(n) &= n+\min\nolimits_k(s(k)+t(n-k)), & v(n) &= n+\min\nolimits_k(u(k)+w(n-k)),\\ t(n) &= n+\min\nolimits_k(u(k)+s(n-k)), & w(n) &= n+\min\nolimits_k(u(k)+v(n-k)). \end{aligned}$$

Отсюда следует, что $r(n)\le s(n)\le u(n)$, $s(n)\le v(n)$ и $r(n)\le t(n)\le w(n)$ для всех n; кроме того, $s(2n)=t(2n+1)=\infty$. (Последнее было очевидно *априори*.)

Пусть $A(n)$ — функция, определяемая правилами $A(1)=0$, $A(2n)=2n+2A(n)$, $A(2n+1)=2n+1+A(n)+A(n+1)$; тогда $A(2n)=2n+A(n-1)+A(n+1)-(0$ или $1)$ для всех $n\ge 2$. Пусть C — константа, такая, что при $4\le n\le 8$

i) для четных n имеем $w(n)\le A(n)+Cn-1$;
ii) для нечетных n имеем $u(n)$ и $v(n)$ оба $\le A(n)+Cn-1$.

(Это в действительности справедливо для всех $C\ge\frac{5}{6}$.) По индукции получаем, что эти соотношения верны для *всех* $n\ge 4$ (в качестве подходящего k выберем $\lfloor n/2\rfloor\pm 1$). Но $A(n)$ является нижней оценкой в (9), если $T=3$ и $r(n)\le\min(u(n),v(n),w(n))$; таким образом, мы доказали, что $A(n)\le\hat{K}_3(n)\le r(n)\le A(n)+\frac{5}{6}n-1$. [Здесь константа $\frac{5}{6}$ может быть уменьшена.]

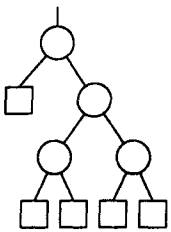
17. [Этот метод использовался в программе сортировки для UNIVAC III и в 1962 году был представлен на симпозиуме ACM Sort Symposium.]

Уровень	T1	T2	T3	T4	T5
0	1	0	0	0	0
1	5	4	3	2	1
2	55	50	41	29	15
. . .	. . .	. . .	. . .	. . .	. . .
n	a_n	b_n	c_n	d_n	e_n
$n+1$	$5a_n+4b_n+3c_n+2d_n+c_n$	$4a_n+4b_n+3c_n+2d_n+e_n$	$3a_n+3b_n+3c_n+2d_n+e_n$	$2a_n+2b_n+2c_n+2d_n+e_n$	$a_n+b_n+c_n+d_n+e_n$

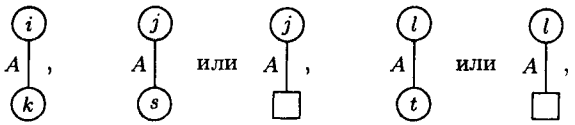
Чтобы перейти с уровня n на уровень $n+1$ во время начального распределения, введите k_1 "подуровней", в которых на ленты (T1, T2, ..., T5) добавляется соответственно $(4,4,3,2,1)$ серий k_2 "подуровней" с $(4,3,3,2,1)$ сериями, k_3 — с $(3,3,2,2,1)$, k_4 — с $(2,2,2,1,1)$, k_5 — с $(1,1,1,1,0)$ сериями, где $k_1 \le a_n$, $k_2 \le b_n$, $k_3 \le c_n$, $k_4 \le d_n$, $k_5 \le e_n$. [Если $(k_1,\ldots,k_5) = (a_n,\ldots,e_n)$, значит, достигнут уровень $n+1$.] Добавьте фиктивные серии, если необходимо дополнить подуровень. Затем выполните слияние $k_1+k_2+k_3+k_4+k_5$ серий с (T1, ..., T5) на T6, $k_1+\cdots+k_4$ серий с (T1, ..., T4) на T5, ..., серии k_1 с T1 на T2, k_1 — с (T2, ..., T6) на T1, k_2 — с (T3, ..., T6) на T2, ... и k_5 — с T6 на T5.

18. (Решение предложено М. С. Патерсоном (M. S. Paterson).) Предположим, запись j помещена в последовательность на ленте номер τ_j. По меньшей мере, $C|\tau|$ записей могут иметь данную последовательность τ, где C зависит от объема внутренней памяти (см. раздел 5.4.8). Следовательно, $|\tau_1|+\cdots+|\tau_N| = \Omega(N\log_T N)$.

19.

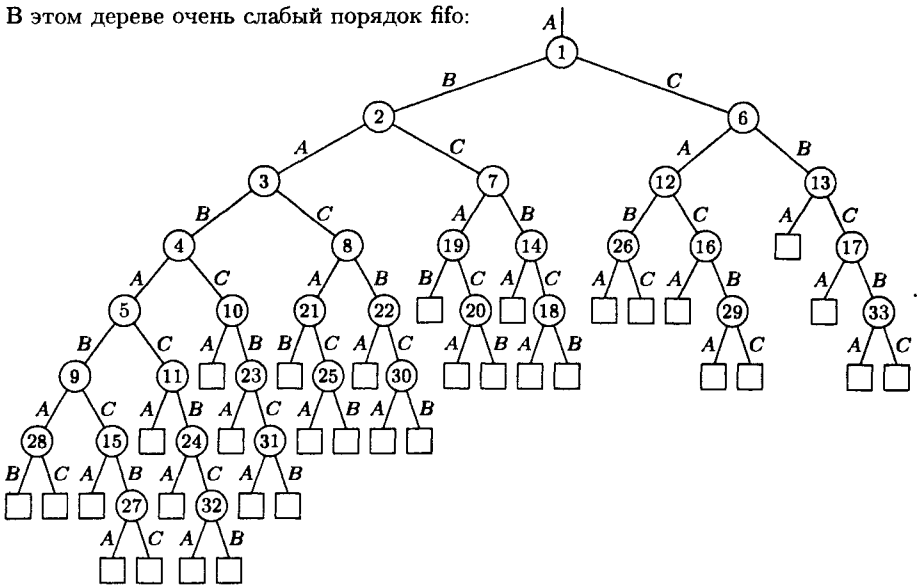


20. Сильное T-fifo-дерево имеет T-fifo-расстановку меток, в которой нет трех ветвей, имеющих вид соответственно

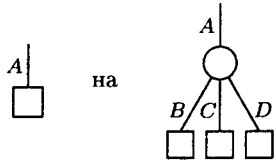


для некоторого имени ленты A и некоторых $i<j<k<l<s$. Неформально, чтобы "вырастить" некоторое A, необходимо вырастить все остальные деревья A до создания какого-либо нового A.

21. В этом дереве очень слабый порядок fifo:



22. Это действительно для любого представления в виде дерева, образуемого последовательной заменой всех вхождений, например заменой



для некоторых фиксированных имен лент A, B, C, D. Так как вхождения заменяются по одному и тому же образцу, порядок LIFO или FIFO не вызывает различий в структуре дерева.

Сформулируем это условие в терминах векторной модели: всякий раз, когда ($y^{(k+1)} \neq y^{(k)}$ или $k = m$) и $y_j^{(k)} = -1$, имеем $y_j^{(k)} + \cdots + y_j^{(1)} + y_j^{(0)} = 0$.

23. (a) Пусть $v_1 \le v_2 \le \cdots \le v_T$; "каскадная" стадия

$$(1, \ldots, 1, -1)^{v_T}(1, \ldots, 1, -1, 0)^{v_{T-1}} \ldots (1, -1, 0, \ldots, 0)^{v_2},$$

которая превращает $C(v)$ в v. (b) Очевидно, так как $C(v)_k \le C(w)_k$ при всех k. (c) Если v получено за q стадий, то имеем $u \to u^{(1)} \to \cdots \to u^{(q)} = v$ для некоторого единичного вектора u и некоторых других элементов $u^{(1)}, \ldots$. Следовательно, $u^{(1)} \preceq C(u)$, $u^{(2)} \preceq C(C(u)), \ldots, v \preceq C^{[q]}(u)$ и $v_1 + \cdots + v_T$ меньше или равно сумме элементов $C^{[q]}(u)$; последнее достигается при каскадном слиянии. [Эта теорема обобщает результат упр. 5.4.3–4; к сожалению, понятие "стадия", как оно определено здесь, не имеет, по-видимому, никакой практической ценности.]

24. Пусть $y^{(m)} \ldots y^{(l+1)}$ будет стадией, которая переводит w в v. Если $y_j^{(i)} = -1$, $y_j^{(i-1)} = 0$, $\ldots$, $y_j^{(k+1)} = 0$ и $y_j^{(k)} = -1$ для некоторого $k < i - 1$, то можно вставить $y^{(k)}$ между $y^{(i)}$ и $y^{(i-1)}$. Повторяем эту операцию, пока все (-1) во всех столбцах не станут соседними. Тогда, если $y_j^{(i)} = 0$ и $y_j^{(i-1)} \neq 0$, можно положить $y_j^{(i)} \leftarrow 1$; в конце концов, все столбцы будут состоять из $+1$, за которыми следуют -1, а за ними — 0. Таким образом, стадия, которая переводит w' в v для некоторого $w' \succeq w$, построена. После перестановки столбцов эта стадия принимает вид $(1, \ldots, 1, -1)^{a_T} \ldots (1, -1, 0, \ldots, 0)^{a_2}(-1, 0, \ldots, 0)^{a_1}$. Последовательность, состоящая из $T - 1$ соотношений

$$\begin{aligned}
(x_1, \ldots, x_T) &\preceq (x_1 + x_T, \ldots, x_{T-1} + x_T, 0) \\
&\preceq (x_1 + x_{T-1} + x_T, \ldots, x_{T-2} + x_{T-1} + x_T, x_T, 0) \\
&\preceq (x_1 + x_{T-2} + x_{T-1} + x_T, \ldots, x_{T-3} + x_{T-2} + x_{T-1} + x_T, x_{T-1} + x_T, x_T, 0) \\
&\preceq \cdots \\
&\preceq (x_1 + x_2 + x_3 + \cdots + x_T, x_3 + \cdots + x_T, \ldots, x_{T-1} + x_T, x_T, 0),
\end{aligned}$$

показывает теперь, что наилучший выбор величин a есть $a_T = v_T$, $a_{T-1} = v_{T-1}$, $\ldots$, $a_2 = v_2$, $a_1 = 0$. Результат является оптимальным, если переставить столбцы так, чтобы выполнялось $v_1 \le \cdots \le v_T$.

25. (a) Предположите, что $v_{T-k+1} \ge \cdots \ge v_T \ge v_1 \ge \cdots \ge v_{T-k}$ и используйте

$$(1, \ldots, 1, -1, 0, \ldots, 0)^{v_{T-k+1}} \ldots (1, \ldots, 1, 0, \ldots, 0, -1)^{v_T}.$$

(b) Для $1 \le l \le T - k$ сумма наибольших l элементов $D_k(v)$ равна $(l-1)s_k + s_{k+l}$. (c) Если $v \Rightarrow w$ в фазе, использующей k выводных лент, то можно, очевидно, считать, что эта фаза имеет вид $(1, \ldots, 1, -1, 0, \ldots, 0)^{a_1} \ldots (1, \ldots, 1, 0, \ldots, 0, -1)^{a_k}$, причем каждая из остальных $T - k$ лент используется как вводная во всех операциях. Выбор $a_1 = v_{T-k+1}$, $\ldots$, $a_k = v_T$

является наилучшим. (d) См. упр. 22, (c). Всегда имеем $k_1 = 1$; $k = T - 2$ всегда лучше, чем $k = T - 1$, так как предполагается, что, по крайней мере, одна компонента вектора v равна нулю. Следовательно, для $T = 3$ имеем $k_1 \ldots k_q = 1^q$ и начальное распределение $(F_{q+1}, F_q, 0)$. Для $T = 4$ найдены следующие недоминирующие стратегии и соответствующие распределения.

$q = 2$ 12 $(3,2,0,0)$
$q = 3$ 121 $(5,3,3,0)$; 122 $(5,5,0,0)$
$q = 4$ 1211 $(8,8,5,0)$; 1222 $(10,10,0,0)$; 1212 $(11,8,0,0)$
$q = 5$ 12121 $(19,11,11,0)$; 12222 $(20,20,0,0)$; 12112 $(21,16,0,0)$.
$q = 6$ 122222 $(40,40,0,0)$; 121212 $(41,30,0,0)$
$q \geq 7$ 12^{q-1} $(5 \cdot 2^{q-3}, 5 \cdot 2^{q-3}, 0, 0)$

Таким образом, для $T = 4$ и $q \geq 6$ слияние с минимальным числом фаз подобно сбалансированному слиянию с незначительными искажениями в самом конце (переход от $(3,2,0,0)$ к $(1,0,1,1)$, а не к $(0,0,2,1)$).

Когда $T = 5$, недоминирующими стратегиями являются $1(32)^{n-1}2$, $1(32)^{n-1}3$ для $q = 2n \geq 2$; $1(32)^{n-1}32$, $1(32)^{n-1}22$, $1(32)^{n-1}23$ для $q = 2n + 1 \geq 3$. (Первая стратегия имеет в своем распределении наибольшее число серий.) На шести лентах они таковы: 13 или 14, 142 или 132 (либо 133), 1333 или 1423 и 13^{q-1} для $q \geq 5$.

РАЗДЕЛ 5.4.5

1. Следующий алгоритм управляется массивом $\mathtt{A}[L-1] \ldots \mathtt{A}[1]\mathtt{A}[0]$, который, в сущности, представляет число, записанное в системе счисления с основанием P. Мы многократно добавляем единицу к этому числу; "переносы" говорят нам, когда следует выполнять слияние. Ленты пронумерованы от 0 до P.

O1. [Начальная установка.] Присвоить $(\mathtt{A}[L-1], \ldots, \mathtt{A}[0]) \leftarrow (0, \ldots, 0)$ и $q \leftarrow 0$. (Во время выполнения этого алгоритма q будет равно $(\mathtt{A}[L-1] + \cdots + \mathtt{A}[0]) \bmod T$.)

O2. [Распределение.] Записать начальную серию на ленту q в порядке возрастания. Установить $l \leftarrow 0$.

O3. [Добавить единицу.] Если $l = L$, остановиться; результат находится на ленте $(-L) \bmod T$ в порядке возрастания тогда и только тогда, когда L четно. В противном случае установить $\mathtt{A}[l] \leftarrow \mathtt{A}[l] + 1$, $q \leftarrow (q+1) \bmod T$.

O4. [Перенос?] Если $\mathtt{A}[l] < P$, то вернуться к шагу O2. В противном случае выполнить слияние на ленту $(q - l) \bmod T$, установить $\mathtt{A}[l] \leftarrow 0$ и $q \leftarrow (q+1) \bmod T$, увеличить l на 1 и вернуться к шагу O3. ▮

2. Следите за числом серий на каждой ленте. Когда исходные данные будут исчерпаны, добавляйте, если необходимо, фиктивные серии, пока не придете к такому положению, когда на каждой ленте будет находиться максимум одна серия и по крайней мере одна лента не будет занята. Затем завершите сортировку посредством еще одного слияния, перемотав сначала некоторые ленты, если это необходимо. (Ориентацию серий можно извлечь из массива A.)

3.

Операция	T0	T1	T2	Операция	T0	T1	T2
Распределение	—	A_1	A_1A_1	Распределение	D_2A_1	A_1	A_4
Слияние	D_2	—	A_1	Слияние	D_2	—	A_4D_2
Распределение	D_2A_1	—	A_1	Слияние	—	A_4	A_4
Слияние	D_2	D_2	—	Распределение	—	A_4	A_4A_1
Распределение	D_2	D_2A_1	A_1	Копирование	—	A_4D_1	A_4
Слияние	D_2D_2	D_2	—	Копирование	—	A_4	A_4A_1
Слияние	D_2	—	A_4	Слияние	D_5	—	A_4

В этот момент T2 можно перемотать — и окончательное слияние завершит сортировку.

Чтобы избежать бесполезных операций копирования, при которых серии просто сдвигаются вперед или назад, можно в конце шага B3 просто сказать "Если исходные данные исчерпаны, перейти к B7" и добавить новый шаг.

B7. [Завершение.] Установить $s \leftarrow -1$ и перейти к B2 после повторения следующей операции до тех пор, пока $l = 0$. Установить $s' \leftarrow \mathtt{A}[l-1, q]$ и установить q' и r' равными индексам, таким, что $\mathtt{A}[l-1, q'] = -1$ и $\mathtt{A}[l-1, r'] = -2$. (Мы получим $q' = r$ и $s' \le \mathtt{A}[l-1, j] \le s'+1$, $j \ne q'$, $j \ne r'$.) Если $s' - s$ нечетно, продвинуть уровень l, в противном случае — откатить его (см. ниже). Затем выполнить слияние на ленте r, читая в обратном порядке; установить $l \leftarrow l-1$, $\mathtt{A}[l, q] \leftarrow -1$, $\mathtt{A}[l, r] \leftarrow s'+1$, $r \leftarrow r'$ и повторить.

Здесь "продвижение" означает повторение следующей операции до тех пор, пока $(q + (-1)^s) \bmod T = r$: установить $p \leftarrow (q + (-1)^s) \bmod T$ и скопировать одну серию с ленты p на ленту q, затем установить $\mathtt{A}[l, q] \leftarrow s+1$, $\mathtt{A}[l, p] \leftarrow -1$, $q \leftarrow p$. "Откат" уровня означает повторение следующей операции до тех пор, пока $(q - (-1)^s) \bmod T = r$: установить $p \leftarrow (q - (-1)^s) \bmod T$ и скопировать одну серию с ленты p на ленту q, затем установить $\mathtt{A}[l, q] \leftarrow s$, $\mathtt{A}[l, p] \leftarrow -1$, $q \leftarrow p$. При операции копирования чтение с ленты p выполняется в обратном направлении, в результате чего направление копируемой серии изменится на противоположное. Если окажется, что при копировании с p на q $D[p] > 0$, то нужно вместо копирования просто уменьшить $D[p]$ и увеличить $D[q]$.

(Основная идея состоит в том, что, если входные данные исчерпаны, желательно исключить, по крайней мере, по одной серии на каждой ленте. Разряд четности каждого неотрицательного элемента $\mathtt{A}[l, j]$ укажет, является ли серия восходящей или нисходящей. Наименьшее S, при котором изменение алгоритма хоть как-то сказывается, есть $P^3 + 1$. Если P велико, такое изменение вряд ли когда-нибудь вызовет большое расхождение, но оно позволит компьютеру не выглядеть слишком глупо при некоторых обстоятельствах. Однако и этот алгоритм можно еще дорабатывать, чтобы более эффективно обрабатывался случай $S = 1$.)

4. На самом деле можно опустить установку $\mathtt{A}[0, 0]$ на шаге B1 и $\mathtt{A}[l, q]$ на шагах B3 и B5. (Однако $\mathtt{A}[l, r]$ *должно* устанавливаться на шаге B3.) Новый шаг B7 в ответе к предыдущему упражнению требует значение $\mathtt{A}[l, q]$ (если только в явном виде не учитывает, что $q' = r$, как отмечается в указанном ответе).

5. $P^{2k} - (P-1)P^{2k-2} < S \le P^{2k}$ при некотором $k > 0$.

РАЗДЕЛ 5.4.6

1. $\lfloor 23000480/(n+480) \rfloor n$.

2. В этот момент все записи правого буфера переданы на вывод. На шаге F2 во время слияния проверка "Полон ли буфер вывода?" предшествует проверке "Пуст ли буфер ввода?"; это существенно, иначе пришлось бы ввести дополнительные проверки (если не внести изменения, как в упр. 4).

3. Нет. Например, мы могли бы достичь состояния, в котором P буферов заполнены на $1/P$ и $P-1$ буферов полны, если бы в файле i находились ключи i, $i+P$, $i+2P$, ... при $1 \le i \le P$. Этот пример показывает, что, даже если разрешить одновременное чтение, необходимо иметь $2P$ буферов ввода для поддержания непрерывного вывода, если только мы не перераспределяем память для частей буферов и не используем каким-либо образом "чтение вразброс". [Вообще говоря, если в блоках содержится меньше $P-1$ записей, требуется меньше $2P$ буферов, но этот случай маловероятен.]

4. Раньше устанавливать S (на шагах F1 и F4, а не F3).

5. Если бы, например, все ключи во всех файлах были равны, то мы не могли бы просто делать произвольный выбор во время прогнозирования; прогноз должен быть совместим с решениями, принимаемыми процессом слияния. Возможный безопасный путь состоит в том, чтобы найти на шагах F1 и F4 наименьшее возможное m, т. е. считать, что все записи из файла $C[i]$ меньше, чем все записи, имеющие тот же ключ в файле $C[j]$, если $i < j$. (В сущности, номер файла присоединяется к ключу.)

6. На шаге C1 установить также $\texttt{TAPE}[T+1] \leftarrow T+1$. На шаге C8 слияние должно идти на $\texttt{TAPE}[p+2]$ вместо $\texttt{TAPE}[p+1]$. На шаге C9 установить $(\texttt{TAPE}[1], \ldots, \texttt{TAPE}[T+1]) \leftarrow (\texttt{TAPE}[T+1], \ldots, \texttt{TAPE}[1])$.

7. Метод, показанный на диаграмме A, есть

$$(A_1D_1)^4A_0D_0(A_1D_1)^2A_0D_0(A_1D_1)^3A_0,$$
$$D_1(A_1D_1)^4A_0D_0(A_1D_1)^3A_0D_0\alpha A_0D_0A_0,$$
$$D_1A_0D_0(A_1D_1)^3A_0D_0\alpha A_1D_1A_0,$$
$$D_1A_1D_1\alpha A_1D_1A_0,$$

где $\alpha = (A_0D_0)^2A_1D_1A_0D_0(A_1D_1)^2(A_0D_0)^7A_1D_1(A_0D_0)^3A_1D_1A_0D_0$. На первой фазе слияния записывается $D_0A_3D_3A_1D_1A_4D_4A_0D_0A_1D_1A_1D_1A_4D_4A_0D_0A_1D_1A_0D_0(A_1D_1)^4$ на ленту 5; на следующей записывается $A_4D_4A_4D_4A_1D_1A_4D_4A_0D_0A_1D_1A_1D_1A_7$ на ленту 1, на следующей — $D_{13}A_4D_4A_0D_0A_{10}$ на ленту 4. Последние фазы таковы:

$A_4D_4A_4$	—	$D_{19}A_3D_3A_{12}$	$D_{13}A_4D_4A_4$	D_0A_3
A_4	$D_{23}A_{11}$	$D_{19}A_3$	$D_{13}A_4$	—
—	D_{23}	D_{19}	D_{13}	D_{22}
A_{77}	—	—	—	—

8. Нет, так как экономится самое большее S стартстопных операций, и, кроме того, время начального распределения, в основном, зависит от скорости обработки вводной ленты (а не выводных лент). Другие преимущества схем распределения, приведенных на диаграмме A, компенсируют этот незначительный недостаток.

9. $P = 5$, $B = 8300$, $B' = 734$, $S = \lceil (3 + 1/P)N/(6P') \rceil + 1 = 74$, $\omega \approx 1.094$, $\alpha \approx 0.795$, $\beta \approx -1.136$, $\alpha' = \beta' = 0$. Значение формулы (9) ≈ 855 с, к которому мы добавляем время начальной перемотки, получая в итоге 958 с. Экономия примерно одной минуты во время слияния не компенсирует потерю времени из-за начальной перемотки и смены ленты (если мы не работаем в мультипрограммном режиме).

10. Во время стандартного многофазного слияния перематывается около 54% файла (столбец "Проходы/фазы" в табл. 5.4.2–1), а самая длинная перемотка в стандартном каскадном слиянии охватывает примерно $a_k a_{n-k}/a_n \approx (4/(2T-1))\cos^2(\pi/(4T-2)) < \frac{4}{11}$ файла (из упр. 5.4.3–5 и соотношения 5.4.3–(13)).

11. Только для начальной и конечной перемоток имеет смысл использовать быструю перемотку, так как бобина, содержащая весь файл примера, заполнена лишь немногим более чем на 10/23. Применяя в примере 8 значения $\pi = \lceil .946 \ln S - 1.204 \rceil$ и $\pi' = 1/8$, получаем следующие итоговые оценки для примеров 1–9:

$$1115,\ 1296,\ 1241,\ 1008,\ 1014,\ 967,\ 891,\ 969,\ 856.$$

12. (a) Очевидное решение с $4P + 4$ буферами состоит в том, чтобы просто одновременно выполнять чтение и запись на спаренные ленты. Заметим, однако, что достаточно трех буферов вывода (в любой момент мы выполняем вторую половину операции записи из одного, первую половину операции записи из второго и заполняем третий). Это наводит на мысль о соответствующем улучшении ситуации с буферами ввода. Оказывается, что

необходимо и достаточно $3P$ буферов ввода и 3 буфера вывода, если использовать несколько ослабленный метод "прогнозирования". Лучший и более простой подход, предложенный Дж. Сю (J. Sue), позволяет добавить к каждому блоку "опережающий ключ", который определяет последний ключ следующего блока. Метод Сю требует $2P+1$ буферов ввода и 4 буфера вывода и является видоизмененным алгоритмом F. (См. также раздел 5.4.9.)

(b) В этом случае большое значение α означает, что число проходов по всем данным, которое мы должны выполнить, лежит между пятью и шестью, что сводит на нет преимущество слияния с двойной скоростью. Эта идея значительно лучше работает на восьми или девяти лентах.

13. Нет. Рассмотрите, например, положение непосредственно перед $A_{16}A_{16}A_{16}A_{16}$. Однако две полные бобины *могут* быть обработаны.

14. $$\det\begin{pmatrix} 0 & -p_0z & 0 & z-1 \\ 0 & 1-p_1z & -p_0z & z-1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \Bigg/ \det\begin{pmatrix} 1-p_{\geq 1}z & -p_0z & 0 & z-1 \\ -p_{\geq 2}z & 1-p_1z & -p_0z & z-1 \\ 0 & -1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

15. Матрица A имеет вид

$$A = \begin{pmatrix} B_{10}z & B_{11}z & \dots & B_{1n}z & 1-z \\ \vdots & & & & \vdots \\ B_{n0}z & B_{n1}z & \dots & B_{nn}z & 1-z \\ 0 & \dots\ 0 & 1 & 0 & 0 \\ 0 & \dots\ 0 & 0 & 0 & 0 \end{pmatrix}, \qquad \begin{matrix} B_{10}+B_{11}+\dots+B_{1n}=1, \\ \vdots \\ B_{n0}+B_{n1}+\dots+B_{nn}=1 \end{matrix}. \tag{11}$$

Следовательно,

$$\det(I-A) = \det\begin{pmatrix} 1-B_{10}z & -B_{11}z & \dots & -B_{1(n-1)}z & -B_{1n}z \\ \vdots & & & & \vdots \\ -B_{n0}z & -B_{n1}z & \dots & 1-B_{n(n-1)}z & -B_{nn}z \\ 0 & 0 & & -1 & 1 \end{pmatrix}$$

и можно прибавить все столбцы к первому, а затем вынести $(1-z)$. Значит, $g_Q(z)$ имеет вид $h_Q(z)/(1-z)$ и $\alpha^{(Q)} = h_Q(1)$, поскольку $h_Q(1) \neq 0$ и $\det(I-A) \neq 0$ для $|z| < 1$.

РАЗДЕЛ 5.4.7

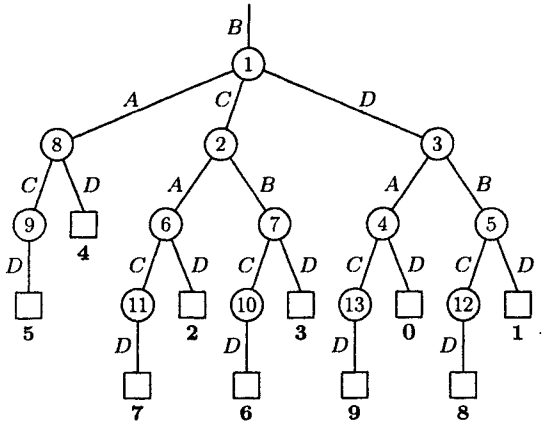
1. Выполняйте сортировку от младших цифр к старшим поочередно в системах счисления с основанниями P и $T-P$. (Если сгруппированы пары цифр, то получим, в сущности, чистое основание $P\cdot(T-P)$. Так, если $P=2$ и $T=7$, то система счисления — двоично-пятеричная, которая очевидным образом связана с десятичными обозначениями.)

2. Если K — ключ, находящийся в диапазоне от 0 до F_n-1, то пусть фибоначчиевым представлением F_n-1-K будет $a_{n-2}F_{n-1}+\dots+a_1F_2$, где a_j равны 0 или 1 и нет двух последовательных единиц. После фазы j на ленте $(j+1) \bmod 3$ находятся ключи с $a_j=0$, а на ленте $(j-1) \bmod 3$ — ключи с $a_j=1$ в порядке убывания значений $a_{j-1}\dots a_1$.

[Представьте сортировальную машину для перфокарт с карманами "0" и "1" и рассмотрите процедуру сортировки F_n карт, на которых перфорированы ключи $a_{n-2}\dots a_1$ в $n-2$ колонках. Обычная процедура их сортировки в порядке убывания, которая начинается с младшей цифры, может быть упрощена, поскольку все, что находится в кармане "1" в конце одного прохода, попадает на следующем проходе в карман "0".]

4. Если бы на уровне 2 находился внешний узел, то нам не удалось бы построить такое хорошее дерево. В противном случае на уровне 3 имеется самое большее три внешних узла, на уровне 4 — шесть, так как предполагается, что каждый внешний узел оказывается на одной и той же ленте.

5.



6. 09, 08, ..., 00, 19, ..., 10, 29, ..., 20, 39, ..., 30, 40, 41, ..., 49, 59, ..., 50, 60, 61, ..., 99.

7. Да. Сначала распределяем записи во все меньшие и меньшие подфайлы, пока не получим файлы на одной бобине, которые могут быть рассортированы отдельно. Этот процесс является двойственным по отношению к сортировке файлов на одной бобине с последующим слиянием их во все большие и большие файлы на нескольких бобинах.

РАЗДЕЛ 5.4.8

1. Да. Пусть направления файла и дерева выбора чередуются от возрастающего к убывающему, тогда в результате получим шейкер-сортировку P-го порядка (см. упр. 9).

2. Пусть $Z_N = Y_N - X_N$; решим рекуррентные соотношения для Z_N, заметив, что

$$(N+1)NZ_{N+1} = N(N-1)Z_N + N^2 + N;$$

следовательно,

$$Z_N = \tfrac{1}{3}(N+1) + \binom{M+2}{3}\Big/N(N-1) \qquad \text{при } N > M.$$

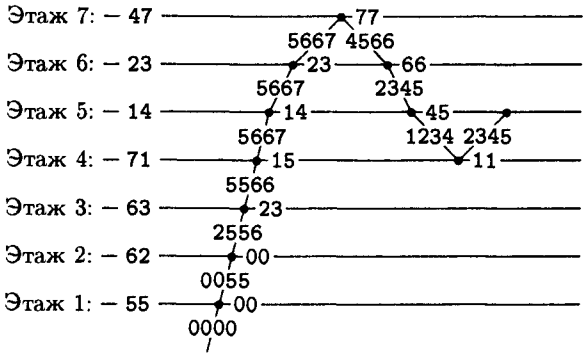
Теперь исключим Y_N и получим

$$\frac{X_N}{N+1} = \frac{20}{3}(H_{N+1} - H_{M+2}) + 2\left(\frac{1}{N+1} - \frac{1}{M+2}\right)$$
$$-\frac{2}{3}\binom{M+2}{3}\left(\frac{1}{(N+1)N(N-1)} - \frac{1}{(M+2)(M+1)M}\right) + \frac{3M+4}{M+2}, \quad N > M.$$

3. Да. Найдите элемент, который является медианой, за $O(N)$ шагов, используя построение, аналогичное приведенному в теореме 5.3.3L, и с его помощью разделите файл. Еще один интересный подход, предложенный Р. У. Флойдом (R. W. Floyd) и Э. Дж. Смитом (A. J. Smith), состоит в том, чтобы слить две серии из N элементов за $O(N)$ единиц времени следующим образом. Раздвиньте элементы на ленте, оставив между ними промежутки, затем последовательно занесите в каждый такой промежуток число, которое определяет конечное положение элемента, предшествующего этому промежутку.

4. Можно объединить график для этажей $\{1, \ldots, p+1\}$ с графиком для этажей $\{q, \ldots, n\}$: когда по первому графику лифт впервые достигает этажа $p+1$, то подняться на этаж q и действовать в соответствии со вторым графиком (считая текущих пассажиров лифта "дополнительными" людьми в алгоритме теоремы K). После выполнения этого графика вернуться на этаж $p+1$ и возобновить прежний график.

5. Рассмотрим $b = 2$, $m = 4$ и следующее поведение алгоритма.



Теперь 2 (в лифте) меньше, чем 3 (на третьем этаже).

[После построения примера, подобного этому, читателю должно быть ясно, как установить справедливость более слабого свойства, используемого в доказательстве теоремы K.]

6. Найдем минимальные i, j, такие, что $b_i < b'_i$ и $b_j > b'_j$. Введем нового человека, который хочет переехать с этажа i на этаж j. При любом k это не увеличит $\max(u_k, d_{k+1}, 1)$ или $\max(b_k, b'_k)$. Продолжим процесс, пока не получим $b_j = b'_j$ при всех j. Теперь заметим, что алгоритм в тексте раздела, если изменить b на b_k на шагах K1 и K3, по-прежнему работает.

8. Пусть их число будет P_n и пусть Q_n — число перестановок, для которых $u_k = 1$ при $1 \le k < n$. Тогда $P_n = Q_1 P_{n-1} + Q_2 P_{n-2} + \cdots + Q_n P_0$, $P_0 = 1$. Можно показать, что $Q_n = 3^{n-2}$ при $n \ge 2$ (см. ниже), поэтому, используя производящие функции, получим

$$\sum P_n z^n = (1 - 3z)/(1 - 4z + 2z^2) = 1 + z + 2z^2 + 6z^3 + 20z^4 + 68z^5 + \cdots;$$

$$2P_n = (2 + \sqrt{2})^{n-1} + (2 - \sqrt{2})^{n-1}.$$

Чтобы доказать, что $Q_n = 3^{n-2}$, рассмотрим тернарную последовательность $x_1 x_2 \ldots x_n$, такую, что $x_1 = 2$, $x_n = 0$ и $0 \le x_k \le 2$ при $1 < k < n$. Следующее правило определяет взаимно однозначное соответствие между такими последовательностями и требуемыми перестановками $a_1 a_2 \ldots a_n$:

$$a_k = \begin{cases} \max\{j \mid (j < k \text{ или } x_j = 0) \text{ или } j = 1\}, & \text{если } x_k = 0; \\ k, & \text{если } x_k = 1; \\ \min\{j \mid (j > k \text{ или } x_j = 2) \text{ или } j = n\}, & \text{если } x_k = 2. \end{cases}$$

(Это соответствие было получено автором совместно с Э. А. Бендером (E. A. Bender).)

9. Число проходов шейкер-сортировки равно $2\max(u_1, \ldots, u_n) - (0$ или $1)$, так как каждая пара проходов (слева направо, справа налево) уменьшает каждое ненулевое u на 1.

10. Начните с какого-нибудь метода распределения (быстрая или обменная поразрядная сортировка), пока не получатся однобобинные файлы. И наберитесь терпения.

РАЗДЕЛ 5.4.9

1. $\frac{1}{4} - (x \bmod \frac{1}{2})^2$ оборотов.

2. Вероятность, что $k = a_{iq}$ и $k + 1 = a_{i'r}$, для фиксированных k, q, r и $i \ne i'$ равна $f(q, r, k) L! L! (PL - 2L)!/(PL)!$, где

$$f(q,r,k) = \binom{k-1}{q-1}\binom{k-q}{r-1}\binom{PL-k-1}{L-q}\binom{PL-k-1-L+q}{L-r}$$
$$= \binom{k-1}{q+r-2}\binom{q+r-2}{q-1}\binom{PL-k-1}{2L-q-r}\binom{2L-q-r}{L-q},$$

и

$$\sum_{\substack{1\le k<PL\\ 1\le q,r\le L}} |q-r| f(q,r,k) = \sum_{1\le q,r\le L} |q-r| \binom{PL-1}{2L-1}\binom{q+r-2}{q-1}\binom{2L-q-r}{L-q} = \binom{PL-1}{2L-1} A_{2L-1}.$$

Вероятность того, что $k = a_{iq}$ и $k+1 = q_{i(q+1)}$, для фиксированных k, q и i равна

$$g(k,q)\Big/\binom{PL}{L}, \quad \text{где} \quad g(k,q) = \binom{k-1}{q-1}\binom{PL-k-1}{L-q-1},$$

и

$$\sum_{\substack{1\le k<PL\\ 1\le q<L}} g(k,q) = \sum_{1\le q<L} \binom{PL-1}{L-1} = (L-1)\binom{PL-1}{L-1}.$$

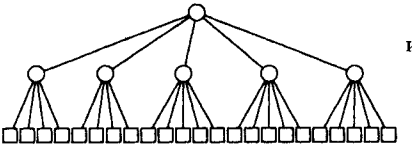
[*SICOMP* **1** (1972), 161–166.]

3. Найдите минимум в соотношении (5) на интервале $2 \le m \le \min(9, n)$.

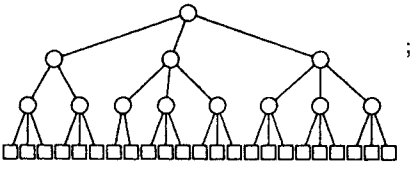
4. (a) $(0.000725(\sqrt{P}+1)^2+0.014)L$. (b) Замените "$\alpha mn+\beta n$" в формуле (5) выражением "$(0.000725(\sqrt{m}+1)^2+0.014)n$". ["Натурные" эксперименты на компьютере показывают, что оптимальные деревья, определяемые этими соотношениями, весьма напоминают деревья, которые определяются теоремой K при $\alpha = 0.00145$, $\beta = 0.01545$. Фактически существуют деревья, оптимальные для обоих рекуррентных соотношений, если $30 \le n \le 100$. Предложенное выше изменение позволяет экономить около 10% времени слияния при условии, что $n = 64$ или 100, как в примере, который рассматривался в основном тексте раздела. Такой способ распределения памяти для буферов рассматривался еще в 1954 году Х. Сьювордом, который показал, что 4-путевое слияние позволяет минимизировать время поиска.]

5. Пусть $A_m(n)$ и $B_m(n)$ — стоимости оптимальных наборов m деревьев, n листьев которых находятся соответственно на четных и нечетных уровнях. Тогда $A_1(1) = 0$, $B_1(1) = \alpha+\beta$; $A_m(n)$ и $B_m(n)$ определены, как в (4), при $m \ge 2$; $A_1(n) = \min_{1\le m\le n}(\alpha mn + \beta n + B_m(n))$, $B_1(n) = \min_{1\le m\le n}(\alpha mn + \beta n + A_m(n))$. Последние уравнения корректны несмотря на то, что $A_1(n)$ и $B_1(n)$ определяются друг через друга!

6.



или



;

$A_1(23) = B_1(23) = 268$. [Интересно отметить, что $n = 23$ есть единственное значение, меньшее или равное 50, при котором никакое дерево с n листьями равной четности не является оптимальным в случае, когда нет ограничений на четность. Возможно, это *единственное* такое значение, когда $\alpha = \beta$.]

7. Для некоторого дерева рассмотрим величины $\alpha d_1 + \beta e_1, \ldots, \alpha d_n + \beta e_n$, где d_j — сумма степеней пути и e_j — есть длина пути для j-го листа. Для оптимального дерева с весами $w_1 \le \cdots \le w_n$ будет выполняться неравенство $\alpha d_1 + \beta e_1 \ge \cdots \ge \alpha d_n + \beta e_n$. Всегда можно переупорядочить индексы так, что $\alpha d_1 + \beta e_1 = \cdots = \alpha d_k + \beta e_k$, где первые k листьев сливаются вместе.

9. Пусть d минимизирует $(\alpha m+\beta)/\ln m$. Рассуждение по индукции с использованием при этом выпуклости показывает, что $A_1(n) \geq (\alpha d+\beta)n\log_d n$, причем равенство наступает при $n = d^t$. Соответствующая верхняя оценка получается из полного d-арного дерева, поскольку в этом случае $D(\tau) = dE(\tau)$, $E(\tau) = tn + dr$ при $n = d^t + (d-1)r$, $0 \leq r \leq d^t$.

10. См. *STOC* **6** (1974), 216–229.

11. Из результата упр. 1.2.4–38 следует $f_m(n) = 3qn+2(n-3^q m)$, когда $2\cdot 3^{q-1} \leq n/m \leq 3^q$; $f_m(n) = 3qn + 4(n-3^q m)$, когда $3^q \leq n/m \leq 2\cdot 3^q$. При этом $f_2(n)+2n \geq f(n)$, причем равенство выполняется тогда и только тогда, когда $4\cdot 3^{q-1} \leq n \leq 2\cdot 3^q$; $f_3(n)+3n = f(n)$; $f_4(n)+4n \geq f(n)$, равенство выполняется тогда и только тогда, когда $n = 4\cdot 3^q$; и $f_m(n)+mn > f(n)$ для всех $m \geq 5$.

12. Используйте спецификации –, 1:1, 1:1:1, 1:1:1:1 or 2:2, 2:3, 2:2:2, ..., $\lfloor n/3\rfloor$:$\lfloor (n+1)/3\rfloor$: $\lfloor (n+2)/3\rfloor$, ...; при этом получаются деревья со всеми листьями на уровне $q+2$, где $4\cdot 3^q \leq n \leq 4\cdot 3^{q+1}$. (Если $n = 4\cdot 3^q$, формируется два таких дерева.)

14. Следующие спецификации деревьев были построены для случаев $n = 1, 2, 3, \ldots$ посредством исчерпывающего просмотра всех разбиений n: –, 1:1, 1:1:1, 1:1:1:1, 1:1:1:1:1, 1:1:1:1:1:1, 1:1:1:1:3, 1:1:3:3, 3:3:3, 1:3:3:3, 3:4:4, 3:3:3:3, 3:3:3:4, 3:3:4:4, 3:4:4:4, 4:4:4:4, ..., 5:6:6:6:12, 6:6:6:6:12, 6:6:6:6:13, ... (По-видимому, степени всегда ≤ 6, но доказать это утверждение довольно сложно.)

15. Если первоначально в лифте находятся a человек, то при первой остановке оценка совместности возрастет не более чем на $a+b$. При следующей остановке оценка возрастет не более чем на $b+m-a$. Значит, после k остановок оценка возрастет не более чем на $kb+(k-1)m$.

16. 11 остановок: **123456** на этаже 2, **334466** на этаже 3, **444666** на этаже 4, **256666** на этаже 5, **466666** на этаже 6, **123445** на этаже 4, **112335** на этаже 5, **222333** на этаже 3, **122225** на этаже 2, **111555** на этаже 5, **111111** на этаже 1. [Это минимум. Результат для 10 остановок при произвольной вместимости лифта может иметь следующий вид. Остановки на этажах: 2, 3, 4, 5, 6, p_2, p_3, p_4, p_5, 1, где $p_2p_3p_4p_5$ есть перестановка множества $\{2,3,4,5\}$. Такой график движения возможен только при $b \geq 8$. См. Martin Gardner, *Knotted Doughnuts* (New York: Freeman, 1986), Chapter 10.]

17. Существует по меньшей мере $(bn)!/b!^n$ конфигураций; число, которое может быть получено из некоторой заданной конфигурации после s остановок, не будет превышать $\left((n-1)\binom{b+m}{b}\right)^{s-1}$, что меньше, чем $(n((b+m)e/b)^b)^s$, полученное в упр. 1.2.6–67. Следовательно, некоторая конфигурация требует

$$s(\ln n + b(1+\ln(1+m/b))) > \ln(bn)! - n\ln b! > bn\ln bn - bn - n((b+1)\ln b - b + 1),$$

как следует из упр. 1.2.5–24.

Замечание. Учитывая то, что $1/(x+y) \geq \frac{1}{2}\min(1/x, 1/y)$, когда x и y положительны, можно выразить эту нижнюю оценку в более приемлемой форме:

$$\Omega\Bigl(\min\Bigl(nb,\ \frac{n\log(1+n)}{\log(1+m/b)}\Bigr)\Bigr).$$

Такой результат получен в работе A. Aggarwal, J. S. Vitter, *CACM* **31** (1988), 1116–1127, в которой также приводится соответствующая верхняя оценка:

$$O\Bigl(\min\Bigl(nb,\ \frac{n\log(1+n)}{\log(1+m/b)}\Bigr)\Bigr).$$

Расширение этого решения для нескольких дисков описано в работе M. H. Nodine, J. S. Vitter, *ACM Symposium on Parallel Algorithms and Architectures* **5** (1993), 120–129.

18. Ожидаемое число остановок есть $\sum_{s\geq 1} p_s$, где p_s — вероятность того, что необходимо не менее s остановок. Пусть $q_s = 1 - p_{s+1}$ — вероятность того, что необходимо не более s остановок. Тогда из упр. 17 следует, что $q_s \leq f(s-1+[s=0])$, где $f(s) = b!^n \alpha^s/(bn)!$ и $\alpha = n((b+m)e/b)^b$. Если $f(t-1) < 1 \leq f(t)$, то $\sum_{s\geq 1} p_s \geq p_1 + \cdots + p_t = t - (q_0 + \cdots + q_{t-1}) \geq t - (f(0) + f(0) + \cdots + f(t-2)) \geq t - (\alpha^{1-t} + \alpha^{1-t} + \cdots + \alpha^{-1}) > t - 1 \geq L - 1$.

19. Осуществляя анализ, выполните шаг (vii) в обратном направлении, распределяя записи сначала в ящик 1, а затем — в ящик 2. Это именно та операция, которая моделируется в отношении файла ключей на шаге (iv). [*Princeton Conference on Information Sciences and Systems* **6** (1972), 140–144.]

20. Следует серьезно отнестись к выбору метода внутренней сортировки, учитывая ограничения, которые накладываются страничной организацией. Такие методы, как метод Шелла с уменьшением смещения, вычисление адреса, пирамидальная сортировка и сортировка списка, вряд ли приемлемы, если мал реальный объем оперативной памяти, поскольку при их реализации потребуется большое число "рабочих страниц". Методы быстрой сортировки, обменной поразрядной сортировки и последовательно распределяемого слияния или поразрядной сортировки значительно лучше "себя чувствуют" в страничной среде.

Существуют определенные нюансы, которые можно учитывать при создании алгоритмов внешней сортировки, но которые исключаются при использовании автоматической системы распределения страниц: (i) выбор на основе предсказания файла исходных данных, который должен читаться следующим, чтобы нужные данные оказались загруженными к тому моменту, когда в них появится необходимость; (ii) выбор размера буфера и порядка слияния в соответствии с характеристиками оборудования и данных.

С другой стороны, виртуальная машина значительно упрощает программирование и с ее помощью можно получить неплохие результаты, если программист работает аккуратно и учитывает возможности той реальной вычислительной системы, для которой разрабатывается программа. Первое достаточно полное исследование этой проблемы выполнено Браун (Brawn), Густавсоном (Gustavson) и Манкиным (Mankin) [*CACM* **13** (1970), 483–494.]

21. $\lceil (L-j)/D \rceil$; см. *CMath*, формула (3.24).

22. После считывания группы из D блоков, которые содержат a_j, понадобится знать α_{j+D-1} прежде, чем будет выполнено считывание следующей группы из D блоков. Если хранить α_{j+D-1} с a_j, то понадобятся также значения $\alpha_0, \ldots, \alpha_{D-2}$ в виде некоторого заголовка файла с тем, чтобы начать процесс.

Но, следуя этой схеме, невозможно записать блоки $a_0 \ldots a_{D-1}$ до тех пор, пока не будет вычислено $a_D \ldots a_{2D-2}$. Таким образом, потребуется $3D-1$ выходных буфера вместо $2D$ для того, чтобы продолжительное время сохранять записываемую информацию. Отсюда следует, что лучше записать значения α в отдельный короткий файл. [Такие же соображения можно привести и по отношению к рандомизированному разделению.]

23. (a) Для реализации алгоритма 5.4.6F требуется 4 входных буфера, каждый объемом, соответствующим размеру суперблока DB. (Если к этому прибавить еще и буфера вывода, то в сумме получится $6DB$ записей в буферах оперативной памяти в случае использования алгоритма 5.4.6F и $5DB$ — в случае использования алгоритма SyncSort.)

(b) В то время, когда считывается группа из D блоков, нужно иметь в своем распоряжении пространство для буферов, чтобы хранить предыдущие D блоков и один незавершенный блок, т. е. $(2D+1)B$ записей. (Для вывода требуется еще $2DB$, но в результате многих операций обработки данных в ходе выполнения 2-путевого слияния на практике передается на выход сравнительно небольшой объем информации.)

24. Пусть l-м блоком в хронологическом порядке будет блок j_l серии k_l, в частности $j_l = 0$ и $k_l = l$ при $1 \le l \le P$. Этот блок будет считываться в момент времени $t_l = \sum_{k=1}^{P} t_{lkd}$, где

$$t_{lkd} = \left|\{r \mid 1 \le r \le l \text{ и } k_r = k \text{ и } (x_k + j_r) \bmod D = d\}\right|$$

есть число блоков серии k на диске d, которые хронологически $\le l$ и $d = (x_{k_l} + j_l) \bmod D$. Пусть $u_{lk} = \left|\{r \mid 1 \le r \le l \text{ и } k_r = k\}\right|$; тогда

$$t_{lkd} = \left\lceil \frac{u_{lk} - (d - x_k) \bmod D}{D} \right\rceil,$$

поскольку j_r пробегает значения $0, 1, \ldots, u_{lk} - 1$, если $1 \le r \le l$ и $k_r = k$. Последовательность t_l для примера из (19)–(21) есть

$$11111\ 22223\ 43456\ 34567\ 82345\ 67893\ \ldots.$$

После того как начнется слияние с l-го в хронологическом порядке блока, количество необходимых блоков в буферах будет составлять $I_l + D + P$, если $l > P$. Здесь I_l есть число "инверсий с равенством" в t_l, а именно — $\left|\{r \mid r > l \text{ и } t_r \le t_l\}\right|$, т. е. число заполненных буферов, в которые мы считали информацию, но еще не готовы ее использовать; D представляет буфера, в которые будут считываться следующие исходные данные, и P — частично заполненные буфера, из которых берутся данные для слияния. (Приняв некоторые специальные меры, можно, используя связи, как в случае с алгоритмом SyncSort, ослабить последнее требование с P до $P - 1$, но усложнение процесса, которое при этом происходит, вряд ли сделает такую операцию целесообразной.)

Таким образом, проблема сведена к определению верхней оценки I_l. Можно предположить, что входные серии имеют бесконечно большую длину. Предположим также, что s в элементах $\{t_1, \ldots, t_l\}$ больше, чем t_l; тогда t_l имеет $t_l D - l + s$ инверсий с равенством, поскольку точно $t_l D$ элементов $\le t_l$. Отсюда следует, что максимум I_l получается, когда $s = 0$ и t_l есть максимум слева направо. Имеем $\sum_{k=1}^{P} u_{lk} = l$; значит, учитывая приведенные выше формулы, для t_l получаем

$$\begin{aligned} I_l \le \max_{l>P}(t_l D - l) &\le \sum_{k=1}^{P}\bigl(u_{lk} - (d - x_k) \bmod D + D - 1 - u_{lk}\bigr) \\ &= P(D-1) - \sum_{k=1}^{P}(d - x_k) \bmod D \\ &\le P(D-1) - \min_{0 \le d < D} \sum_{k=1}^{P}(d - x_k) \bmod D \end{aligned}$$

и существуют хронологические порядки, для которых достижима эта верхняя оценка.

Предположим, что r_t из x_k равны t. Желательно выбрать x_k таким образом, чтобы максимизировать $\min_{0 \le d < D} s_d$, где $s_d = \sum_{k=1}^{P}(d - x_k) \bmod D = \sum_{t=0}^{D-1}((d - t) \bmod D) r_t$. Можно предположить, что минимум будет при $d = 0$. Тогда $s_1 = s_0 + P - r_1 D$, $s_2 = s_1 + P - r_2 D, \ldots$. Отсюда получаем $r_1 \le \lfloor P/D \rfloor$, $r_1 + r_2 \le \lfloor 2P/D \rfloor, \ldots$, поэтому минимумом является

$$s_0 = (D-1)r_1 + (D-2)r_2 + \cdots + r_{D-1} \le \sum_{k=1}^{D-1} \lfloor kP/D \rfloor = \frac{1}{2}\bigl((P-1)(D-1) + \gcd(P, D) - 1\bigr),$$

что вытекает из результатов упр. 1.2.4–37. Эта граница достигается, когда $x_j = \lceil jD/P \rceil - 1$ при $1 \le j \le P$.

Имея такое x_j, можно работать с любой хронологической последовательностью на полной скорости, если в нашем распоряжении есть $I_{\max} + D + P = \frac{1}{2}PD + \frac{3}{2}D + \frac{1}{2}P +$

$\frac{1}{2}\gcd(P,D)-1$ входных буферов, в каждом из которых отведено место для B записей. (Это особенно хорошо, когда $D=2$ или 3.)

25. Обратите внимание на то, что в цикле 4 мы возвращаемся к чтению f_1 с диска 0.

	Активные чтения	Активные слияния	Прочие	В ожидании
Цикл 1	$e_0b_0g_0a_0c_0$	$-----$	$(-----)$	a_0
Цикл 2	$f_1d_0d_1d_2f_0$	a_0-----	$b_0c_0(e_0g_0---)$	d_0
Цикл 3	$a_2h_0e_2g_1d_3$	$a_0b_0c_0d_0----$	$e_0f_0g_0(d_1d_2f_1-)$	h_0
Цикл 4	$f_1e_1b_1g_1a_1$	$a_0b_0c_0d_0e_0f_0g_0h_0$	$d_1(d_2e_2d_3f_1g_1a_2)$	e_1
Цикл 5	$a_2f_2h_1e_3g_2$	$a_0b_0c_0d_1e_1f_0g_0h_0$	$d_2e_2d_3a_1f_1b_1g_1()$	a_2
Цикл 6	$d_4a_3f_3b_2e_4$	$a_2b_1c_0d_3e_2f_1g_1h_0$	$f_2e_3(h_1g_2---)$	d_4
Цикл 7	$c_1a_3f_3\ ?\ e_4$	$a_2b_1c_0d_4e_3f_2g_1h_0$	$(h_1b_2g_2a_3f_3e_4-)$	c_1
Цикл 8	$?\ d_5d_6\ ?\ ?$	$a_2b_1c_1d_4e_3f_2g_1h_0$	$h_1b_2g_2a_3f_3e_4(\ ?\ )$	d_5

26. В то время, когда D блоков считываются и D блоков записываются, процедура слияния могла бы сформировать до $P+Q-1$ блоков на выход, предполагая выполнение схемы (24). (Но не $P+Q$, поскольку только один буфер слияния будет абсолютно пуст.) Считывание выполняется с той же скоростью, что и запись, а потому необходимо $D+P+Q-1$ буферов вывода для предотвращения задержки вывода.

Однако в среднем не более D блоков являются выводными для каждых входных D блоков, так что на практике нужно ориентироваться примерно на $3D$ выходных буферов.

27. (a) $E_n(m_1,\ldots,m_p)=\sum_{t=1}^{m}q_t$, где q_t есть вероятность того, что в некоторой урне содержится не менее t шаров. Очевидно, что $q_t\le 1$ и

$$q_t\le\sum_{k=0}^{n-1}\Pr(\text{урна } k \text{ содержит не менее } t \text{ шаров})=n\Pr(S_n(m_1,\ldots,m_p)\ge t).$$

(b) Производящая функция вероятностей для $S_n(m_1,\ldots,m_p)$ такова:

$$p(z)=\prod_{k=1}^{p}z^{q_k}(1+(z-1)r_k/n),$$

где $q_k=\lfloor m_k/n\rfloor$ и $r_k=m_k \bmod n$. Теперь $1+\alpha\le(1+\alpha/n)^n$ и $1+\alpha r/n\le(1+\alpha/n)^r$, когда $\alpha\ge 0$; отсюда имеем $\Pr(S_n(m_1,\ldots,m_p)\ge t)\le(1+\alpha)^{-t}p(1+\alpha)\le(1+\alpha)^{-t}\prod_{k=1}^{p}(1+\alpha/n)^{m_k}=(1+\alpha)^{-t}(1+\alpha/n)^m$.

Если $t\le m/n$, используется член "1" в сформулированном минимуме. Если $t>m/n$, величина $(1+\alpha)^{-t}(1+\alpha/n)^m$ принимает минимальное значение $(n-1)^{m-t}m^m/(n^mt^t(m-t)^{m-t})$, когда $\alpha=(nt-m)/(m-t)$.

28. Как нам кажется, числовой подсчет подтверждает это предположение. Например, получим

$$\begin{array}{rrr}
E_{10}(1,1,1,1,1,1,1,1)=2.3993180, & E_{10}(2,2,2,2)=2.178, & E_{10}(4,3,1)=2.00,\\
E_{10}(2,1,1,1,1,1,1)=2.364540, & E_{10}(3,2,2,1)=2.166, & E_{10}(5,2,1)=1.98,\\
E_{10}(2,2,1,1,1,1)=2.32076, & E_{10}(3,3,1,1)=2.152, & E_{10}(6,1,1)=1.94,\\
E_{10}(3,1,1,1,1,1)=2.29958, & E_{10}(4,2,1,1)=2.138, & E_{10}(4,4)=1.7,\\
E_{10}(2,2,2,1,1)=2.2628, & E_{10}(5,1,1,1)=2.090, & E_{10}(5,3)=1.7,\\
E_{10}(3,2,1,1,1)=2.2460, & E_{10}(3,3,2)=2.02, & E_{10}(6,2)=1.7,\\
E_{10}(4,1,1,1,1)=2.2076, & E_{10}(4,2,2)=2.01, & E_{10}(7,1)=1.7.
\end{array}$$

29. (a) В момент t со всех дисков считываются блоки, которые появятся не ранее, чем блок, маркированный в момент t. Следующие Q блоков никогда не будут удалены из группы прочих буферов, если они прочитаны. Таким образом, интересующие нас блоки на

диске j будут считаны во время $\le t + N_j$; все они должны принять участие в слиянии во время $t + \max(N_0, \ldots, N_{D-1})$.

(b) Если $(Q+1)$-й блок после маркированного блока не удален, то можно использовать тот же аргумент. В противном случае предыдущие Q не маркированы и $Q+2$ блоков не могут размещаться на разных дисках.

(c) Разделите хронологический порядок блоков на группы размером $Q+2$ и рассмотрите любую из них. Если существует M_k блоков из серии k, то числа N_j эквивалентны числу шаров в j-й урне в задаче о циклической занятости при $n = D$ и $m = Q+2$. Таким образом, ожидаемое число маркированных ячеек не превышает верхней оценки в упр. 27, (b). Обозначим эту верхнюю оценку через $e_n(m)$. Тогда $r(d,m) = (d/m)e_d(m)$.

[В действительности такая функция $r(2,m)$ не монотонна по m, когда m мало́. Таким образом, элементы, перечисленные в табл. 2 для $r(2,4)$ и $r(2,12)$, в действительности являются значениями $r(2,3)$ и $r(2,11)$. Дополнительные буфера не могут увеличить число маркированных блоков.]

30. Пусть $l = \lceil (s+\sqrt{2s})\ln d \rceil$, $\alpha = \sqrt{2/s}$. Тогда

$$\begin{aligned} e_d(sd\ln d) &< l + \sum_{t>l} d(1+\alpha/d)^{sd\ln d}/(1+\alpha)^t \\ &= l + d(1+\alpha/d)^{sd\ln d}/\alpha(1+\alpha)^l \\ &\le l + \alpha^{-1}\exp((\ln d)(1+s\alpha-(s+\sqrt{2s})\ln(1+\alpha))) \end{aligned}$$

и $(s+\sqrt{2s})\ln(1+\alpha) > s\alpha + 1 - \alpha/3$. Таким образом,

$$1 \le r(d, sd\ln d) = \frac{e_d(sd\ln d)}{s\ln d} < 1 + \sqrt{\frac{2}{s}} + \frac{1}{\sqrt{2s}\ln d}\left(1 + \sqrt{\frac{2}{9s}}\ln d + O(s^{-1}(\log d)^2)\right),$$

если $s/(\log d)^2 \to \infty$. Сходимость этого асимптотического выражения довольно слабая (см. табл. 2).

31. Когда $Q = 0$, маркируется первый блок и затем периодически маркируется следующий блок, который разделяет диск с одним из тех блоков, которые находятся в группе, начинающейся с ранее маркированного блока. Например, если хронологический порядок обращения к диску — 112020121210122, маркировка будет иметь вид $\bar{1}\bar{1}20\bar{2}01\bar{2}1\bar{2}10\bar{1}2\bar{2}$. Таким образом, при $P \to \infty$ считывается в среднем $Q(D)n$ блоков в течение n тактов времени, где Q — функция Раманьяна, определенная в соотношении 1.2.11.3-(2). И напротив, $r(d,2) = (d+1)/2$ дает значительно более пессимистическую оценку.

РАЗДЕЛ 5.5

1. *Трудно решить, какой алгоритм сортировки наилучший в данной ситуации.* ▌

2. При малых N наилучшим будет метод вставки в список; при средних значениях N, например при $N = 64$, — метод слияния списков; при больших N — метод поразрядной сортировки списка.

3. (Решение В. Пратта (V. Pratt).) Пусть заданы две неубывающие серии α и β и их нужно слить. Определим очевидным способом подсерии $\alpha_1\alpha_2\alpha_3\beta_1\beta_2\beta_3$, такие, что α_2 и β_2 содержат в точности ключи из α и β, имеющие медианное значение всего массива. Выполнив "перебрасывание" подсерий, сформируем сначала $\alpha_1\alpha_2\beta_1^R\alpha_3^R\beta_2\beta_3$, затем — $\alpha_1\beta_1\alpha_2^R\beta_2^R\alpha_3\beta_3$ и, наконец, — $\alpha_1\beta_1\alpha_2\beta_2\alpha_3\beta_3$. В результате можно свести задачу к слиянию подмассивов $\alpha_1\beta_1$ и $\alpha_3\beta_3$, имеющих длину $\le N/2$.

Значительно более сложный алгоритм предложен Л. Трабб Пардо (L. Trabb Pardo). Этот алгоритм обеспечивает наилучший из возможных результат в асимптотическом смысле. Можно выполнять устойчивое слияние за время порядка $O(N)$ и сортировать за

время порядка $O(N \log N)$, используя только $O(\log N)$ бит дополнительной памяти для фиксированного числа индексных переменных, причем не требуется никакого специального преобразования исходных данных [см. *SICOMP* **6** (1977), 351–372]. Те же самые показатели по времени выполнения и объему памяти получены в работе B.-C. Huang, M. A. Langston, *Comp. J.* **35** (1992), 643–650. См. также A. Symvonis, *Comp. J.* **38** (1995), 681–690, где описан метод устойчивого слияния M элементов в N, если M значительно меньше, чем N.

4. Только методы простой вставки, вставки в список и слияния списков. Некоторые варианты метода быстрой сортировки также можно сделать бережливыми, но только ценой дополнительных операций во внутреннем цикле (см. упр. 5.2.2–24).

Особое значение приобретают бережливые методы в том случае, когда результат сравнения обладает надежностью на все 100% (см. D. E. Knuth, *Lecture Notes in Comp. Sci.* **606** (1992), 61–67).

РАЗДЕЛ 6.1

1. $\sqrt{(N^2-1)/12}$; см. уравнение 1.2.10–(22).

2. **S1′.** [Инициализация.] Установить P ← FIRST.

S2′. [Сравнение.] Если $K =$ KEY(P), алгоритм успешно завершается.

S3′. [Продвижение.] Установить P ← LINK(P).

S4′. [Конец файла?] Если P $\neq \Lambda$, перейти к шагу S2′. В противном случае алгоритм завершается неудачно. ▌

3.

```
KEY     EQU  3:5
LINK    EQU  1:2
START   LDA  K            1
        LD1  FIRST        1
2H      CMPA 0,1(KEY)     C
        JE   SUCCESS      C
        LD1  0,1(LINK)    C − S
        J1NZ 2B           C − S
FAILURE EQU  *            1 − S  ▌
```

Время выполнения равно $(6C - 3S + 4)u$.

4. Да, если можно установить KEY(Λ) равным K. (Впрочем, метод дублирования цикла из программы Q′ в этом случае не даст никаких преимуществ.)

5. Нет, программа Q всегда выполняет не меньше операций, чем программа Q′.

6. Замените команды строки 08 командами `JE *+4`; `CMPA KEY+N+2,1`; `JNE 3B`; `INC1 1`, а команды строк 03 и 04 — командами `ENT1 -2-N`; `3H INC1 3`.

7. Заметьте, что $\bar{C}_N = \frac{1}{2}\bar{C}_{N-1} + 1$.

8. Формула суммирования Эйлера дает

$$H_n^{(x)} = \zeta(x) + \frac{n^{1-x}}{(1-x)} + \frac{1}{2}n^{-x} - \frac{B_2 x}{2!}n^{-1-x} + \frac{B_3 x(x+1)}{3!}n^{-2-x} - O(n^{-3-x}).$$

(Из теории функций комплексных переменных известно, что

$$\zeta(x) = 2^x \pi^{x-1} \sin(\tfrac{1}{2}\pi x)\Gamma(1-x)\zeta(1-x).$$

Эта формула наиболее полезна при $x < 0$.)

9. (a) Да: $\bar{C}_N = N - N^{-\theta} H_{N-1}^{(-\theta)} = N + 1 - N^{-\theta} H_N^{(-\theta)} = \frac{\theta}{1+\theta} N + \frac{1}{2} + O(N^{-\theta})$.

(b) $\bar{C}_N = \frac{\theta}{1+\theta}\left(1 + N/\left(1 - \binom{N-\theta}{N}\right)\right) = \frac{\theta}{1+\theta}(N + N^{1-\theta}/\Gamma(1-\theta) + 1) + O(N^{1-2\theta})$.

(c) При $\theta < 0$ (11) не является распределением вероятностей; (16) дает оценку $\bar{C}_N = -\frac{\theta}{1+\theta}\Gamma(1-\theta)N^{1+\theta} + O(N^{1+2\theta}) + O(1)$ вместо (15).

10. $p_1 \le \cdots \le p_N$; (максимальное $\bar{C}_N$) $= (N+1) -$ (минимальное $\bar{C}_N$). (Аналогично в записях с переменной длиной максимальное среднее время поиска равно $L_1(1+p_1) + \cdots + L_N(1+p_N)$ минус минимальное среднее время поиска.)

11. (a) Произведения $f_{m-1}(x_{i_1}, \ldots, x_{i_{m-1}})p_i$ представляют собой вероятности возможных последовательных запросов, после выполнения которых элемент R_i оказывается на m-м месте. (b) Второе тождество получается после суммирования всех $\binom{n}{m}$ вариантов первого подмножества на различных m-подмножествах X с учетом того, что каждый из них встречается P_{nk} раз. Третье тождество является результатом обращения второго. (Можно также использовать принцип включения и исключения.) (c) $\sum_{m\ge 0} mP_{nm} = nQ_{nn} - Q_{n(n-1)}$, а потому

$$d_i = 1 + (N-1) - p_i \sum_{j\ne i} \frac{1}{p_i + p_j};$$

$$\sum_i p_i d_i = N - \sum_{i<j} \frac{p_i^2 + p_j^2}{p_i + p_j} = N - \sum_{i<j}\left(p_i + p_j - \frac{2p_ip_j}{p_i+p_j}\right) = (17).$$

Примечание. В. Дж. Хендрикс (W. J. Hendricks) [*J. Applied Probability* **9** (1972), 231–233] нашел простую формулу для финальных вероятностей каждой перестановки записей. Например, при $N = 4$ предельная вероятность последовательности $R_3\,R_1\,R_4\,R_2$ равна

$$\frac{p_3}{p_3+p_1+p_4+p_2}\;\frac{p_1}{p_1+p_4+p_2}\;\frac{p_4}{p_4+p_2}\;\frac{p_2}{p_2}.$$

Джеймс Битнер (James Bitner) [*SICOMP* **8** (1979), 82–85] доказал, что, если изначально список неупорядочен, ожидаемое время поиска после t случайных запросов превысит $\widetilde{C}_N$ на величину $\frac{1}{4}\sum_{i,j}(p_i-p_j)^2(1-p_i-p_j)^t/(p_i+p_j)$. Таким образом, для t поисков потребуется в среднем менее $t\widetilde{C}_N + \frac{1}{4}\sum_{i,j}(p_i-p_j)^2/(p_i+p_j)^2 < t\widetilde{C}_N + \frac{1}{2}\binom{N}{2}$ сравнений. См. также доказательство с применением производящих функций в работе P. Flajolet, D. Gardy, and L. Thimonier, *Discrete Applied Math.* **39** (1992), 207–229, §6.

12. $\widetilde{C}_N = 2^{1-N} + 2\sum_{n=0}^{N-2} 1/(2^n+1)$. Это выражение быстро сходится к $2\alpha' \approx 2.5290$; в упр. 5.2.4–13 дано значение α' с точностью до 40 знаков.

13. После выполнения весьма утомительного суммирования

$$\sum_{k=1}^{n} k^2 H_{n+k} = \frac{n(n+1)(2n+1)}{6}(2H_{2n} - H_n) - \frac{n(n+1)(10n-1)}{36}$$

получим ответ:

$$\widetilde{C}_N = \tfrac{4}{3}N - \tfrac{2}{3}(2N+1)(H_{2n} - H_n) + \tfrac{5}{6} - \tfrac{1}{3}(N+1)^{-1} \approx .409N.$$

14. В предположении, что $x_1 \le x_2 \le \cdots \le x_n$, максимальное значение достигается при $y_{a_1} \le y_{a_2} \le \cdots \le y_{a_n}$ и минимальное — при $y_{a_1} \ge \cdots \ge y_{a_n}$. Рассуждения аналогичны рассуждениям при доказательстве теоремы S.

15. Рассуждая так же, как в теореме S, можно прийти к выводу, что расположение $R_1R_2\ldots R_N$ оптимально тогда и только тогда, когда

$$P_1/L_1(1-P_1) \ge \cdots \ge P_N/L_N(1-P_N).$$

16. Ожидаемое время проверки $T_1+p_1T_2+p_1p_2T_3+\cdots+p_1p_2\cdots p_{N-1}T_N$ минимально тогда и только тогда, когда $T_1/(1-p_1) \le \cdots \le T_N/(1-p_N)$. [*BIT* **3** (1963), 255–256; некоторые интересные дополнения получены Джеймсом Р. Слейглом (James R. Slagle), *JACM* **11** (1964), 253–264.]

17. Выполняйте задания в порядке увеличения их крайних сроков выполнения — независимо от значений T_j! (Естественно, в реальной жизни одни задания важнее других и требуется минимизировать максимальное *взвешенное* запаздывание; возможно, придется минимизировать сумму $\sum_{i=1}^{n} \max(T_{a_1}+\cdots+T_{a_i} - D_{a_i}, 0)$. Похоже, однако, что простого решения для таких постановок задач не существует.)

18. Положим $h = 1$ при наличии s и $h = 0$ в противном случае. Пусть $A = \{j \mid q_j < r_1\}$, $B = \{j \mid q_j = r_j\}$, $C = \{j \mid q_j > r_j\}$, $D = \{j \mid t_j > 0\}$; тогда сумма $\sum_{i,j} p_ip_jd_{|i-j|}$ для расположения (q, r) минус соответствующая сумма для расположения (q', r') будет равна

$$2\sum_{i\in A,\, j\in C}(q_i-r_i)(q_j-r_j)(d_{|i-j|}-d_{h+1+2k-i-j})+2\sum_{i\in C,\, j\in D}(q_i-r_i)t_j(d_{h+2k-i+j}-d_{i-1+j}).$$

Эта величина положительна, за исключением случаев, когда $C = \emptyset$ или $A\cup D = \emptyset$. Искомый результат следует из того, что расположения в виде "органных труб" — единственные расположения, которые не могут быть улучшены таким способом (и с помощью его зеркального двойника) при $m = 0$, 1.

(Этот результат получен Г. Г. Харди (G. H. Hardy), Дж. Э. Литлвудом (J. E. Littlewood) и Д. Пойа (G. Pólya) [*Proc. London Math. Soc.* (2), **25** (1926), 265–282], которые показали, что минимум $\sum_{i,j} p_iq_jd_{|i-j|}$ достигается для всех независимых перестановок p и q только при "органном порядке" p и q. Более подробные разъяснения и обобщения можно найти в их книге *Inequalities* (Cambridge University Press, 1934), Chapter 10.)

19. Все расположения одинаково хороши. Считая, что $d(j, j) = 0$, находим

$$\sum_{i,j} p_ip_j\, d(i,j) = \tfrac{1}{2}\sum_{i,j} p_ip_j\big(d(i,j)+d(j,i)\big) = \tfrac{1}{2}c.$$

(Частный случай $d(i, j) = 1 + (j - i) \bmod N$ для $i \ne j$ был рассмотрен К. Ю. Айверсоном (K. E. Iverson) в работе *A Programming Language* (New York: Wiley, 1962), 138. Р. Л. Бейбер (R. L. Baber) в работе *JACM* **10** (1963), 478–486, изучил некоторые задачи, связанные с поиском на ленте с возможностью чтения, перемотки или возврата на k блоков без чтения. В. Д. Фрейзер (W. D. Frazer) нашел, что возможно значительное ускорение поиска в случае репликации в файле некоторой информации. В работе E. B. Eichelberger, W. C. Rodgers, and E. W. Stacy, *IBM J. Research & Development* **12** (1968), 130–139, имеется эмпирическое решение схожей задачи.)

20. Переходя, как и в упр. 18, от (q, r) к (q', r'), получим изменение

$$\sum_{i\in A,\, j\in C}(q_i-r_i)(q_j-r_j)\big(d_{|i-j|}-\min(d_{h+1+2k-i-j}, d_{i+j-1})\big),$$

которое всегда положительно, кроме случаев, когда A или C является пустым множеством. Вследствие циклической симметрии оптимальные перестановки представляют собой циклические сдвиги "органной" конфигурации. (О другом классе задач с тем же ответом можно прочесть в работе T. S. Motzkin and E. G. Straus, *Proc. Amer. Math. Soc.* **7** (1956), 1014–1021.)

21. Эта задача впервые была решена Л. Х. Харпером (L. H. Harper) [*SIAM J. Appl. Math.* **12** (1964), 131–135]. Обобщения и ссылки на другие работы можно найти в *J. Applied Probability* **4** (1967), 397–401.

22. Приоритетная очередь размером 1 000 элементов (представленная, например, в виде кучи; см. раздел 5.2.3). Вставьте в очередь первые 1 000 записей, причем элементы с *бо́льшими* значениями $d(K_j, K)$ вставляются в начало очереди. Затем каждым последующим K_j, для которого $d(K_j, K) < d(\text{начало очереди}, K)$, следует заменить первый элемент очереди и переупорядочить ее.

РАЗДЕЛ 6.2.1

1. Докажите по индукции, что при достижении шага B2 $K_{l-1} < K < K_{u+1}$ и что $l \le i \le u$ при достижении шага B3.

2. (a, c) Нет; алгоритм зациклится при $l = u - 1$ и $K > K_u$. (b) Да. Однако при отсутствии в таблице K он будет зацикливаться при $l = u$ и $K < K_u$.

3. Это алгоритм 6.1T при $N = 3$. При успешном завершении поиска алгоритм выполняет в среднем $(N + 1)/2$ сравнений; в противном случае среднее число сравнений составляет $N/2 + 1 - 1/(N + 1)$.

4. Это должен быть неудачный поиск с $N = 127$; следовательно, согласно теореме B ответ равен $138u$.

5. Среднее время работы программы 6.1Q′ составляет $1.75N + 8.5 - (N \bmod 2)/4N$; таким образом, она опережает программу B тогда и только тогда, когда $N \le 44$. (Программа C проигрывает при $N \le 11$.)

7. (a) Определенно, нет. (b) Примечания в скобках в алгоритме U остаются справедливыми, а потому алгоритм будет работать, но только если при нечетном N ключи $K_0 = -\infty$ и $K_{N+1} = +\infty$ будут присутствовать в таблице.

8. (a) N. Интересно доказать это по индукции, заметив, что при замене N на $N + 1$ увеличивается ровно одно из приращений δ. (В *AMM* **77** (1970), 884, можно найти обобщение этого утверждения.) (b) Максимум $= \sum_j \delta_j = N$; минимум $= 2\delta_1 - \sum_j \delta_j = N \bmod 2$.

9. Тогда и только тогда, когда $N = 2^k - 1$.

10. Используйте "макрорасширение" программы, содержащей таблицу DELTA. Так, для $N = 10$ получим следующее.

```
START  ENT1  5
       LDA   K
       CMPA  KEY,1
       JL    C3A
C4A    JE    SUCCESS        C3A  EQU   *
       INC1  3                     DEC1  3
       CMPA  KEY,1                 CMPA  KEY,1
       JL    C3B                   JGE   C4B
C4B    JE    SUCCESS        C3B  EQU   *
       INC1  1                     DEC1  1
       CMPA  KEY,1                 CMPA  KEY,1
       JL    C3C                   JGE   C4C
C4C    JE    SUCCESS        C3C  EQU   *
       INC1  1                     DEC1  1
       CMPA  KEY,1                 CMPA  KEY,1
       JE    SUCCESS               JE    SUCCESS
       JMP   FAILURE               JMP   FAILURE  ▌
```

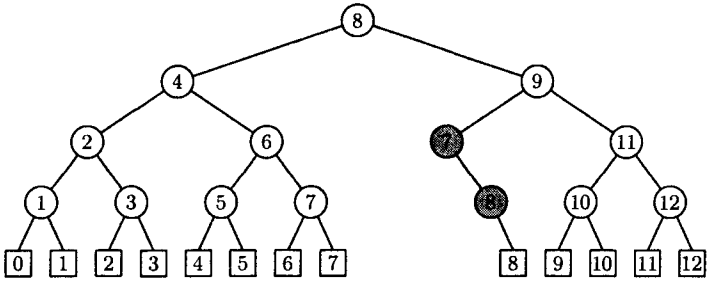
(В упр. 23 показано, что большинство команд JE можно исключить, получив в результате программу из примерно $6 \lg N$ строк, которая выполняется за около $4 \lg N$ единиц времени. Однако такая программа окажется быстрее только для $N \ge 1000$ (приблизительно).)

11. Рассмотрим соответствующее дерево (например, такое, как на рис. 6): при нечетном N левое поддерево корня представляет собой зеркальное отражение правого поддерева и $K < K_i$ встречается так же часто, как и $K > K_i$. В среднем, $C1 = \frac{1}{2}(C+S)$ и $C2 = \frac{1}{2}(C-S)$, $A = \frac{1}{2}(1-S)$. При четном N дерево имеет тот же вид, что и при $N+1$, с метками, уменьшенными на 1 (за исключением узла 0, который при этом становится излишним). В среднем, полагая $k = \lfloor \lg N \rfloor$, имеем:

$$C1 = \frac{C+1}{2} - \frac{k}{2N}, \qquad C2 = \frac{C-1}{2} + \frac{k}{2N}, \qquad A = 0 \qquad \text{при } S = 1;$$
$$C1 = \frac{(k+1)N}{2(N+1)}, \qquad C2 = \frac{(k+1)(N+2)}{2(N+1)}, \qquad A = \frac{N}{2(N+1)} \qquad \text{при } S = 0.$$

(Среднее значение C указано в тексте.)

12.



13.

$N =$	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
$C_N =$	1	$1\frac{1}{2}$	$1\frac{2}{3}$	$2\frac{1}{4}$	$2\frac{1}{5}$	$2\frac{2}{6}$	$2\frac{3}{7}$	$3\frac{1}{8}$	3	3	3	$3\frac{2}{12}$	$3\frac{3}{13}$	$3\frac{3}{14}$	$3\frac{4}{15}$	$4\frac{1}{16}$
$C'_N =$	1	$1\frac{2}{3}$	2	$2\frac{3}{5}$	$2\frac{4}{6}$	3	3	$3\frac{6}{9}$	$3\frac{6}{10}$	$3\frac{8}{11}$	$3\frac{8}{12}$	4	4	4	4	$4\frac{13}{17}$

14. Одна идея заключается в поиске наименьшего $M \geq 0$, такого, что $N + M$ имеет вид $F_{k+1} - 1$. Затем следует установить $i \leftarrow F_k - M$ на шаге F1 и вставить в начало шага F2 "Если $i \leq 0$, то перейти к шагу F4". Лучшим решением будет адаптация метода Шара к поиску Фибоначчи: если результат самого первого сравнения $K > K_{F_k}$, установить $i \leftarrow i - M$ и перейти к шагу F4 (далее все продолжает работать, как обычно). Это позволяет избежать дополнительных затрат времени работы во внутреннем цикле.

15. Внешние узлы появляются на уровнях от $\lfloor k/2 \rfloor$ до $k-1$; разница между этими числами превышает единицу для всех k, кроме $k = 0, \ldots, 4$.

16. Дерево Фибоначчи порядка k, отображенное зеркально (так что левое и правое поддеревья меняются местами), согласно "естественному соответствию" из раздела 2.3.2 представляет собой диаграмму размножения кроликов по k-й месяц включительно (надо только удалить верхний узел диаграммы).

17. Пусть длина пути равна $k - A(n)$; тогда $A(F_j) = j$ и $A(F_j + m) = 1 + A(m)$ при $0 < m < F_{j-1}$.

18. Успешный поиск: $A_k = 0$, $C_k = (3kF_{k+1} + (k-4)F_k)/5(F_{k+1} - 1) - 1$, $C1_k = C_{k-1}(F_k - 1)/(F_{k+1} - 1)$. Неудачный поиск: $A'_k = F_k/F_{k+1}$, $C'_k = (3kF_{k+1} + (k-4)F_k)/5F_{k+1}$, $C1'_k = C'_{k-1}F_k/F_{k+1} + F_{k-1}/F_{k+1}$. $C2 = C - C1$. (См. также упр. 1.2.8–12.)

20. (a) $b = p^{-p}q^{-q}$. (b) В приведенном рассуждении, по меньшей мере, *две* ошибки. Первая состоит в том, что деление не является линейной функцией и его нельзя просто "усреднять". В самом деле, с вероятностью p получим pN элементов, с вероятностью q их будет qN, так что математическое ожидание составляет $(p^2 + q^2)N$; следовательно, искомый множитель равен $1/(p^2 + q^2)$. Теперь, когда мы выяснили, что после k итераций

множитель равен $1/(p^2+q^2)^k$, нельзя утверждать, что $b = 1/(p^2+q^2)$, поскольку количество итераций, необходимых для поиска одних элементов, гораздо больше, чем требуется для поиска других. Это и есть вторая ошибка. (Остерегайтесь подобных ловушек, так как зачастую в теории вероятностей ошибочные утверждения выглядят очень правдоподобно!)

21. Это невозможно, так как метод зависит от величины ключа.

22. *FOCS* **17** (1976), 173–177. См. также Y. Perl, A. Itai, and H. Avni, *CACM* **21** (1978), 550–554; G. H. Gonnet, L. D. Rogers, and J. A. George, *Acta Informatica* **13** (1980), 39–52; G. Louchard, *RAIRO Inform. Théor.* **17** (1983), 365–385; *Computing* **46** (1991), 193–222. Отклонение составляет $O(\log\log N)$. Широкомасштабные эмпирические исследования Д. Марсальи (G. Marsaglia) и Б. Нарасимхана (B. Narasimhan) [*Computers and Math.* **26**, 8 (1993), 31–42] показали, что среднее число обращений к таблице очень близко к $\lg\lg N$, плюс около 0.7 при неудачном поиске. При $N = 2^{20}$, например, для случайного успешного поиска требуется около 4.29 обращений, в то время как для неудачного — около 5.05.

23. При $\geq$ следует идти вправо, при $<$ — влево; по достижении узла $\boxed{i}$, как следует из (1), выполняется $K_i \leq K < K_{i+1}$ и последняя проверка покажет успешность проведенного поиска. (В таблице должен присутствовать ключ $K_0 = -\infty$.)

В алгоритме C может быть изменен шаг C2, на котором можно переходить к шагу C4, если выполняется условие $K = K_i$. На шаге C3 в случае, если `DELTA[j]` $= 0$, установите $i \leftarrow i-1$ и перейдите к шагу C5. На шаге C4, если `DELTA[j]` $= 0$, также переходите к шагу C5, который должен выглядеть следующим образом: "При $K = K_i$ алгоритм завершается успешно, в противном случае алгоритм завершается неудачно". (Такое изменение может ускорить программу C только при $N > 2^{26}$; среднее время поиска при внесении изменений "уменьшается" с $(8.5\lg N - 6)u$ до $(8\lg N + 7)u$.)

24. Ключи могут располагаться таким образом, что сначала будет установлено $i \leftarrow 1$, затем — $i \leftarrow 2i$ или $2i+1$ в зависимости от того, $K < K_i$ или $K > K_i$. При $i > N$ поиск неудачен. Например, при $N = 12$ расположение ключей должно быть следующим:

$$K_8 < K_4 < K_9 < K_2 < K_{10} < K_5 < K_{11} < K_1 < K_{12} < K_6 < K_3 < K_7.$$

Время работы такой программы на `MIX`-компьютере примерно равно $6\lg N$ единицам. Таким образом, эта программа работает быстрее программы C, однако для начальной установки таблицы требуется известная доля труда и сообразительности.

25. (a) Поскольку $a_0 = 1 - b_0$, $a_1 = 2a_0 - b_1$, $a_2 = 2a_1 - b_2$ и т. д., имеем $A(z) + B(z) = 1 + 2zA(z)$. Несколько формул, приведенных в разделе 2.3.4.5, можно вывести из этого соотношения, рассматривая $A(1)$, $B(1)$, $B(\frac{1}{2})$, $A'(1)$ и $B'(1)$. При использовании двух переменных для того, чтобы отличить шаги влево от шагов вправо, получим более общий результат $A(x,y) + B(x,y) = 1 + (x+y)A(x,y)$, представляющий собой частный случай формулы, справедливой для t-арных деревьев (см. R. M. Karp, *IRE Transactions* **IT-7** (1961), 27–38).

(b) $\operatorname{var}(g) = ((N+1)/N)\operatorname{var}(h) - ((N+1)/N^2)\operatorname{mean}(h)^2 + 2$.

26. Дерево для трехленточного многофазного слияния, соответствующее распределению с заполненным уровнем k, есть дерево Фибоначчи порядка $k+1$; надо только поменять правую и левую ветви. (Перерисуйте дерево на рис. 76 из раздела 5.4.4, поменяв местами левые и правые поддеревья A и C. При этом должен получиться аналог рис. 8).

27. Поскольку можно упорядочить индексы таким образом, что $K_{i_1} < K_{i_2} < \cdots < K_{i_k}$, возможно максимум $k+1$ исходов из 2^k. Таким образом, поиск может быть описан с помощью дерева с узлами, из которых выходит не более $(k+1)$ ветвей. Количество записей, которые могут быть найдены на m-м шаге, — не более $k(k+1)^{m-1}$. Следовательно, среднее

количество сравнений, как минимум, равно сумме N наименьших элементов мультимножества $\{k\cdot 1, k(k+1)\cdot 2, k(k+1)^2\cdot 3, \ldots\}$, умноженной на N^{-1}. При $N \geq (k+1)^n - 1$ среднее количество сравнений $\geq ((k+1)^n - 1)^{-1} \sum_{m=1}^{n} k(k+1)^{m-1} m > n - 1/k$.

28. [*Skrifter udgivne af Videnskabs-Selskabet i Christiania*, Mathematisk-Naturvidenskabelig Klasse (1910), No. 8; перепечатано в Thue's *Selected Mathematical Papers* (Oslo: Universitetsforlaget, 1977), 273–310]. (a) T_n имеет $F_{n+1} + F_{n-1} = F_{2n}/F_n$ листьев. (Это так называемое число Лукаса $L_n = \phi^n + \hat{\phi}^n$.) (b) Аксиома гласит, что $T_0(T_2(x)) = T_1(x)$, и получается $T_m(T_n(x)) = T_{m+n-1}(x)$ при $m = 1$ или $n = 1$. По индукции по n результат справедлив при $m = 0$; например, $T_0(T_3(x)) = T_0(T_2(x)*T_1(x)) = T_0(T_1(T_2(x))*T_0(T_2(x))) = T_0(T_2(T_2(x))) = T_2(x)$. И на последнем шаге следует использовать индукцию по m.

29. Пусть $K_0 = -\infty$ и $K_{N+1} = K_{N+2} = \infty$. Сначала проведем бинарный поиск на множестве $K_2 < K_4 < \cdots$; это потребует максимум $\lfloor \lg N \rfloor$ сравнений. Если поиск неудачен, при этом определяется интервал $K_{2j-2} < K < K_{2j}$; K отсутствует, если $2j = N + 2$. В противном случае бинарный поиск для K_{2j-1} определит i, такое, что $K_{2i-2} < K_{2j-1} < K_{2i}$. Теперь может быть два исхода — либо $K = K_{2i-1}$, либо K отсутствует. (См. *Theor. Comp. Sci.* **58** (1988), 67.)

30. Пусть $n = \lfloor N/4 \rfloor$. Начиная с $K_1 < K_2 < \cdots < K_N$, можно расположить K_1, K_3, $\ldots$, K_{2n-1} в любом требуемом порядке, поменяв их местами с любой перестановкой из $K_{2n+1}, K_{2n+3}, \ldots, K_{4n-1}$; такое расположение удовлетворяет условиям предыдущего упражнения. Теперь пусть $K_1 < K_3 < \cdots < K_{2^{t+1}-3}$ — границы между всеми возможными t-битовыми числами. Вставим $K_{2^{t+1}-1}, K_{2^{t+1}+1}, \ldots, K_{2^{t+1}+2m-3}$ между этим "частоколом" в соответствии со значениями $x_1, x_2, \ldots, x_m$. Например, если $m = 4$, $t = 3$, $x_1 = (001)_2$, $x_2 = (111)_2$ и $x_3 = x_4 = (100)_2$, требуемый порядок таков:

$$K_1 < K_{15} < K_3 < K_5 < K_7 < K_{19} < K_{21} < K_9 < K_{11} < K_{13} < K_{17}.$$

(Допускается также, чтобы K_{21} предшествовало K_{19}.) Бинарный поиск $K_{2^{t+1}+2j-3}$ в подмножестве $K_1 < K_3 < \cdots < K_{2^{t+1}-3}$ будет эквивалентен поиску битов числа x_j слева направо. (См. Fiat, Munro, Naor, Schäffer, Schmidt, and Siegel, *J. Comp. Syst. Sci.* **43** (1991), 406–424.)

РАЗДЕЛ 6.2.2

1. Используйте головной узел, скажем, с ROOT ≡ RLINK(HEAD); начните выполнять алгоритм с шага T4 с P ← HEAD. Шаг T5 должен выполняться так, как будто $K >$ KEY(HEAD). (Соответственно в программе T следует заменить команды строк 04 и 05 командами "ENT1 ROOT; CMPA K").

2. На шаге T5 установите RTAG(Q) ← 1. Кроме того, при вставке влево установите RLINK(Q) ← P, при вставке вправо установите RLINK(Q) ← RLINK(P) и RTAG(P) ← 0. На шаге T4 измените проверку "RLINK(P) $\neq \Lambda$" на "RTAG(P) $\neq 0$". (Если узлы вставляются в последовательные ячейки памяти Q и если все удаления осуществляются по принципу LIFO ("последним вошел — первым вышел"), поля RTAG могут быть удалены, поскольку RTAG(P) будет равно 1 тогда и только тогда, когда RLINK(P) < P. Подобное примечание справедливо и для левой, и для правой прошивок.)

3. Можно заменить Λ корректным адресом и установить KEY(Λ) ← K в начале работы алгоритма. В таком случае проверки LLINK и RLINK = Λ могут быть удалены из внутреннего цикла. Однако для корректного выполнения вставки необходимо ввести другой указатель, следующий за P. Это можно сделать, дублировав код так же, как и в программе 6.2.1F. Работа программы при этом не замедляется. Время работы такой модифицированной программы уменьшается до $5.5C$ единиц.

4. $C_N = 1 + (0\cdot 1 + 1\cdot 2 + \cdots + (n-1)2^{n-1} + C'_{2^n-1} + \cdots + C'_{N-1})/N = (1+1/N)C'_N - 1$ при $N \ge 2^n - 1$. Решением этих уравнений является $C'_N = 2(H_{N+1} - H_{2^n}) + n$, $N \ge 2^n - 1$. При этом экономится $2H_{2^n} - n - 2 \approx n(\ln 4 - 1)$ сравнений. Улучшение для $n = 1, 2, 3, 4$ составляет соответственно 0, $\frac{1}{6}$, $\frac{61}{140}$, $\frac{274399}{360360}$; для малых n выигрыш, как видите, невелик. (См. статью Frazer and McKellar, *JACM* **17** (1970), 502, в которой подробно описана эквивалентная задача сортировки.)

5. (a) Первым должен быть элемент **CAPRICORN**; затем умножаем количество способов построения левого поддерева на количество способов построения правого поддерева и на $\binom{10}{3}$, количество совместных перестановок этих двух последовательностей. Таким образом, искомый ответ равен

$$\binom{10}{3}\binom{2}{0}\binom{1}{0}\binom{0}{0}\binom{6}{3}\binom{2}{0}\binom{1}{0}\binom{0}{0}\binom{2}{1}\binom{0}{0}\binom{0}{0} = 4800.$$

(В общем случае ответ представляет собой произведение $\binom{l+r}{r}$ по всем узлам, где l и r означают размеры левого и правого поддеревьев этого узла. Данное произведение равно $N!$, деленному на произведение размеров поддеревьев. В результате получилась та же формула, что и в упр. 5.1.4–20; в самом деле, существует очевидное однозначное соответствие между перестановками, позволяющими получить некоторое дерево поиска, и "топологическими" перестановками, подсчитанными в указанном упражнении — при замене a_k в дереве поиска на k (с использованием формы записи из упр. 6).) (b) $2^{N-1} = 1024$; на каждом шаге, кроме последнего, должен вставляться либо наименьший, либо наибольший из оставшихся ключей.

6. (a) Для каждой из P_{nk} перестановок $a_1 \ldots a_{n-1}a_n$, стоимость которых равна k, построим $n+1$ перестановку $a'_1 \ldots a'_{n-1} m\, a'_n$, где $a'_j = a_j$ или $a_j + 1$ в зависимости от выполнения условия $a_j < m$ или $a_j \ge m$ (см. раздел 1.2.5, метод 2). Если $m = a_n$ или $a_n + 1$, такая перестановка имеет цену $k+1$; в противном случае ее цена равна k. (b) $G_n(z) = (2z + n - 2)(2z + n - 3)\ldots(2z)$. Поэтому

$$P_{nk} = \left[{n-1 \atop k}\right] 2^k.$$

Эта производящая функция была получена, в сущности, В. Ч. Линчем (W. C. Lynch) [*Comp. J.* **7** (1965), 299–302]. (c) Производящая функция для вероятностей есть $g_n(z) = G_n(z)/n!$. Она является произведением простых производящих функций вероятностей, а потому

$$\operatorname{var}(g_n) = \sum_{k=0}^{n-2} \operatorname{var}\left(\frac{2z+k}{2+k}\right) = \sum_{k=0}^{n-2}\left(\frac{2}{k+2} - \frac{4}{(k+2)^2}\right) = 2H_n - 4H_n^{(2)} + 2.$$

(С помощью упр. 6.2.1–25, (b) можно вычислить среднее значение и дисперсию C'_n, чтобы вычислить дисперсию C_n, которая равна $(2 + 10/n)H_n - 4(1 + 1/n)(H_n^{(2)} + H_n^2/n) + 4$; эта формула выведена Г. Д. Кноттом (G. D. Knott).)

7. Сравнение с k-м наибольшим элементом будет выполнено тогда и только тогда, когда этот элемент появится до m-го и до всех элементов между k- и m-м; вероятность этого равна $1/(|m-k|+1)$. Суммируя по k, получим искомый ответ: $H_m + H_{n+1-m} - 1$ [*CACM* **12** (1969), 77–80; см. также L. Guibas, *Acta Informatica* **4** (1975), 293–298].

8. (a) $g_n(z) = z^{n-1}\sum_{k=1}^{n} g_{k-1}(z)g_{n-k}(z)/n$, $g_0(z) = 1$.

(b) $7n^2 - 4(n+1)^2H_n^{(2)} - 2(n+1)H_n + 13n$. (В статье P. F. Windley, *Comp. J.* **3** (1960), 86, приведены рекуррентные соотношения, по которым можно найти дисперсию численно, однако в ней не содержится решение проблемы. Обратите внимание на *отсутствие* простой связи с дисперсией C_n, найденной в ответе к упр. 6.)

10. Например, каждое слово x ключа можно заменить словом $ax \bmod m$, где m — размер машинного слова, а a — случайный множитель, взаимно простой с m. Можно также порекомендовать величину, близкую к $(\phi - 1)m$ (см. раздел 6.4). Гибкое распределение памяти при работе с деревьями может сделать такие методы более притягательными, чем схемы с хешированием.

11. $N - 2$; однако это происходит с вероятностью $1/(N\,N!)$ только при удалении

$$\textcircled{1}\ N\ N{-}1\ \ldots\ 2.$$

12. $\frac{1}{2}(n+1)(n+2)$ удалений в доказательстве теоремы H относятся к случаю 1, так что ответ — $(N+1)/2N$.

13. Да. Доказательство теоремы H показывает, что если удалить k-й вставленный элемент, то для любого фиксированного k результат будет случайным. (Г. Д. Кнотт (Ph. D. thesis, Stanford, 1975) показал, что результат остается случайным после любой фиксированной последовательности вставок и удалений элементов $(k_1, \ldots, k_d)$.)

14. Пусть `NODE(T)` находится на уровне k и пусть `LLINK(T)` $= \Lambda$, `RLINK(T)` $= \mathtt{R}_1$, `LLINK(`$\mathtt{R}_1$`)` $= \mathtt{R}_2, \ldots,$ `LLINK(`$\mathtt{R}_d$`)` $= \Lambda$, где $\mathtt{R}_d \neq \Lambda$ и $d \geq 1$. Допустим, что в правом поддереве узла `NODE(`$\mathtt{R}_i$`)` находится n_i, $1 \leq i \leq d$, внутренних узлов. При наличии шага D$1\frac{1}{2}$ длина внутреннего пути уменьшается на $k + d + n_1 + \cdots + n_d$; без этого шага она уменьшается на $k + d + n_d$.

15. 11, 13, 25, 11, 12. (Если a_j является наименьшим, средним, наибольшим из $\{a_1, a_2, a_3\}$, после удаления дерево будет получено $(4, 2, 3) \times 4$ раз.)

16. Да; коммутативной будет даже операция удаления из перестановок, определенная в доказательстве теоремы H (если пренебречь перенумерацией). При наличии элемента между элементами X и Y коммутативность удаления очевидна, поскольку на операцию удаления влияет только относительное расположение элементов X, Y и их наследников и удаления X и Y не взаимодействуют между собой. С другой стороны, если Y является преемником X и Y — больший елемент, то оба порядка удаления просто удаляют элементы X и Y. Если Y — преемник X, а Z — преемник Y, оба удаления приведут к замещению *первого* встреченного элемента X, Y или Z элементом Z и к удалению второго и третьего из встретившихся элементов из перестановки.

18. Воспользуйтесь упр. 1.2.7–14.

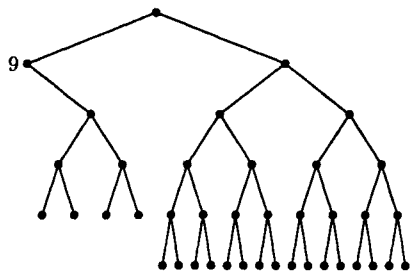
19. $2H_N - 1 - 2\sum_{k=1}^{N}(k-1)^\theta/kN^\theta = 2H_N - 1 - 2/\theta + O(N^{-\theta})$. (Распределение Парето 6.1–(13) приводит к подобному асимптотическому результату с поправкой $O(n^{-\theta}\log n)$.)

20. Да, конечно. Предположим, что $K_1 < \cdots < K_N$, так что дерево, построенное при помощи алгоритма T, вырожденное. Если, например, $p_k = (1 + ((N+1)/2 - k)\epsilon)/N$, среднее число сравнений равно $(N+1)/2 - (N^2-1)\epsilon/12$, в то время как оптимальное дерево требует менее $\lceil \lg N \rceil$ сравнений.

21. $\frac{1}{8}, \frac{3}{20}, \frac{9}{20}, \frac{3}{20}, \frac{1}{8}$. (Большинство углов равны 30°, 60° или 90°.)

22. При $d = 2$ это очевидно; при $d > 2$ имеем $r[i, j-1] \leq r[i+1, j-1] \leq r[i+1, j]$.

23.



(Увеличение веса первого узла в конце концов приведет к его перемещению в корневое положение. Это наводит на мысль о сложности динамического поддержания оптимальности дерева.)

24. Пусть c означает цену дерева, полученного посредством удаления n-го узла из оптимального дерева. Тогда $c(0, n-1) \le c \le c(0, n) - q_{n-1}$, поскольку операция удаления всегда перемещает $\boxed{n-1}$ на один уровень вверх. Аналогично $c(0, n) \le c(0, n-1) + q_{n-1}$, так как при предлагаемой замене цена дерева равна последнему выражению, откуда следует, что $c(0, n-1) = c = c(0, n) - q_{n-1}$.

25. (a) Предположим, что $A \le B$ и $B \le C$, и пусть $a \in A$, $b \in B$, $c \in C$, $c < a$. Если $c \le b$, то $c \in B$; следовательно, $c \in A$ и $a \in B$; значит, $a \in C$. Если $c > b$, то $a \in B$; следовательно, $a \in C$ и $c \in B$; значит, $c \in A$. (b) Это легко доказать самостоятельно.

26. Цена любого дерева имеет вид $y + lx$, где $y \ge 0$ — некоторое действительное число, а l — целое, большее 0. Минимум конечного числа таких функций (взятых по всем деревьям) всегда имеет описанный вид.

27. (a) Из ответа к упр. 24 (в частности, из того, что $c = c(0, n-1)$) вытекает, что $R(0, n-1) = R(0, n) \setminus \{n\}$.

(b) Если $l = l'$, результат в указании тривиален. В противном случае обозначим пути к $\boxed{n}$ как

$$r_0, r_1, \ldots, \boxed{r_l} \quad \text{и} \quad s_0, s_1, \ldots, \boxed{s_{l'}}.$$

Поскольку $r = r_0 > s_0 = s$ и $r_{l'} < s_{l'} = n$, можно найти уровень $k \ge 0$, такой, что $r_k > s_k$ и $r_{k+1} \le s_{k+1}$. По индукции имеем $r_{k+1} \in R(r_k, n)$, $s_{k+1} \in R(s_k, n)$ и $R(s_k, n) \le R(r_k, n)$. Значит, $r_{k+1} \in R(s_k, n)$ и $s_{k+1} \in R(r_k, n)$; отсюда следует результат, приведенный в указании.

Теперь для доказательства того, что $R'_h \le R_h$, положим, что $r \in R'_h$, $s \in R_h$, $s < r$, и рассмотрим оптимальные деревья, показанные для $x = x_h$. Получим, что $l \ge l_h$, и можно предположить, что $l' = l_h$. Для доказательства соотношения $R_h \le R'_{h+1}$ положим, что $r \in R_h$, $s \in R'_{h+1}$, $s < r$, и рассмотрим оптимальные деревья, показанные для $x = x_{h+1}$. Получим, что $l' \le l_h$, и можно предположить, что $l = l_h$.

29. Это вырожденное дерево (см. упр. 5) с `YOU` на вершине дерева и `THE` внизу, которое требует в среднем 19.158 сравнений.

Дуглас А. Гамильтон (Douglas A. Hamilton) доказал, что наихудшим деревом всегда является некоторое вырожденное дерево. Таким образом, существует $O(n^2)$-алгоритм получения наихудших бинарных деревьев поиска.

30. См. R. L. Wessner, *Information Processing Letters* **4** (1976), 90–94; F. F. Yao, *SIAM J. Algebraic and Discrete Methods* **3** (1982), 532–540.

31. См. *Acta Informatica* **1** (1972), 307–310.

32. Когда M достаточно велико, оптимальное дерево должно иметь указанный вид и минимальная цена должна быть в M раз больше минимальной длины внешнего пути плюс решение сформулированной проблемы.

(*Примечание.* Статья Весснера, указанная в ответе к упр. 30, поясняет, как найти оптимальное бинарное дерево поиска высоты $\le L$. Для частного случая $p_1 = \cdots = p_n = 0$ решение было получено Т. Ч. Ху (T. C. Hu) и К. Ч. Таном (K. C. Tan) (MRC Report 1111 (Univ. of Wisconsin, 1970)). А. М. Гарсия (A. M. Garsia) и М. Л. Воч (M. L. Wachs) доказали, что в этом случае все внешние узлы окажутся максимум на двух уровнях, если $\min_{k=1}^{n}(q_{k-1} + q_k) \ge \max \sum_{k=0}^{n} q_k$, а также представили алгоритм, требующий только $O(n)$ шагов для поиска оптимального двухуровневого дерева).

33. Решение поставленной задачи можно найти в A. Itai, *SICOMP* **5** (1976), 9–18; альтернативные варианты рассмотрены в работе D. Spuler, *Acta Informatica* **31** (1994), 729–740.

34. Согласно аппроксимации Стирлинга при $p_1 \ldots p_n \neq 0$ это значение равно $2^{H(p_1,\ldots,p_n)N} \times$ $\times (2\pi N)^{(1-n)/2}(p_1 \ldots p_n)^{-1/2}(1+O(1/N))$.

35. Минимальное значение правой части неравенства, равное $1-p+H(p,1-p)$, достигается при $2x = (1-p)/p$. Однако согласно (20) при $k=2$ $H(p,q,r) \leq 1-p+H(p,1-p)$.

36. Сначала докажем утверждение из указания, принадлежащее Дженсену (Jensen) (*Acta Math.* **30** (1906), 175–193). Если f вогнута, следовательно, функция $g(p) = f(px+(1-p)y) - pf(x) - (1-p)f(y)$ вогнута; кроме того, она удовлетворяет условию $g(0) = g(1) = 0$. Если для некоторого $0 < p < 1$ $g(p) < 0$, то должны существовать $p_0 < p$, для которого $g'(p_0) < 0$, и $p_1 > p$, для которого $g'(p_1) > 0$ согласно теореме о среднем. Однако это противоречит условию вогнутости. Следовательно, $f(px+(1-p)y) \geq pf(x) + (1-p)f(y)$ при $0 \leq p \leq 1$, что геометрически очевидно. Теперь по индукции можно доказать, что $f(p_1x_1 + \cdots + p_nx_n) \geq p_1f(x_1) + \cdots + p_nf(x_n)$, поскольку $f(p_1x_1 + \cdots + p_nx_n) \geq p_1f(x_1) + \cdots + p_{n-2}f(x_{n-2}) + (p_{n-1}+p_n)f((p_{n-1}x_{n-1}+p_nx_n)/(p_{n-1}+p_n))$ при $n > 2$.

Согласно лемме E имеем

$$H(XY) = H(X) + \sum_{i=1}^{m} p_i H(r_{i1}/p_i, \ldots, r_{in}/p_i);$$

и последняя сумма $\sum_{j=1}^{n}\sum_{i=1}^{m} p_i f(r_{ij}/p_i) \leq \sum_{j=1}^{n} f(\sum_{i=1}^{m} r_{ij}) = H(Y)$, где функция $f(x) = x\lg(1/x)$ вогнута.

37. Согласно п. (a) упр. 3.3.2–26 имеем $\Pr(P_1 \geq s) = (1-s)^{n-1}$. Отсюда

$$\mathrm{E}\,H(P_1,\ldots,P_n) = n\,\mathrm{E}\,P_1\lg(1/P_1) = n\int_0^1 (1-s)^{n-1}\,d(s\lg(1/s)) = -(A+B)/\ln 2,$$

где $A = n\int_0^1 (1-s)^{n-1}\,ds = 1$ и

$$B = n\int_0^1 (1-s)^{n-1}\ln s\,ds = \sum_{k=1}^{n}\binom{n}{k}(-1)^k s^k\left(\frac{1}{k} - \ln s\right)\Bigg|_0^1 = -H_n$$

согласно упр. 1.2.7–13. Таким образом, ответ равен $(H_n - 1)/\ln 2$. (Это значение $\lg n + (\gamma - 1)/\ln 2 + O(n^{-1})$ очень близко к максимальной энтропии $H(\frac{1}{n},\ldots,\frac{1}{n}) = \lg n$, и $H(p_1,\ldots,p_n)$ с высокой вероятностью равна $\Omega(\log n)$.)

38. Если $s_{k-1} = s_k$, имеем $q_{k-1} = p_k = q_k = 0$ (см. (26)). Постройте дерево для $n-1$ вероятности $(p_1,\ldots,p_{k-1},p_{k+1},\ldots,p_n; q_0,\ldots,q_{k-1},q_{k+1},\ldots,q_n)$ и замените лист $\boxed{k\text{–}1}$ двухлистовым поддеревом.

39. Можно провести доказательство так же, как и в теореме M, если $0 < w_1 \leq w_2 \leq \cdots \leq w_n$ и $s_k = w_1 + \cdots + w_k$, поскольку из $w_k \geq 2^{-t}$ следует, что $s_{k-1} + 2^{-t} \leq s_k \leq s_{k+1} - 2^{-t}$ при упорядоченных весах. Следовательно, имеем $|\sigma_k| < 1 + \lg(1/w_k)$. (Этот результат вместе с соответствующей нижней границей $H(w_1,\ldots,w_n)$ составлял теорему 9 в оригинальной статье Шеннона 1948 года.)

40. При $k = s+3$ указанная перестановка изменяет цену с $q_{k-1}l + q_kl + q_{k-2}l_{k-2}$ до $q_{k-2}l + q_{k-1}l + q_kl_{k-2}$, так что изменение равно $(q_{k-2} - q_k)(l - l_{k-2})$; эта величина отрицательна при $l < l_{k-2}$, поскольку $q_{k-2} > q_k$.

Точно так же при $k \geq s+4$ перестановка изменяет цену на величину

$$\begin{aligned}\delta = {} & q_{s+1}(l - l_{s+1}) + q_{s+2}(l - l_{s+2}) + q_{s+3}(l_{s+1} - l_{s+3}) + \cdots + q_{k-2}(l_{k-4} - l_{k-2}) \\ & + q_{k-1}(l_{k-3} - l) + q_k(l_{k-2} - l).\end{aligned}$$

Мы имеем $q_{s+1} > q_{s+3}$, $q_{s+2} > q_{s+4}$, $\ldots$, $q_{k-2} > q_k$. Таким образом, находим

$$\delta \leq (q_{k-2} - q_k)(l - l_{k-2}) + (q_{k-3} - q_{k-1})(l - l_{k-3}) \leq 0;$$

например, при четном $k-s$

$$\delta \le q_{k-3}(l-l_{s+1}) + q_{k-2}(l-l_{s+2}) + q_{k-3}(l_{s+1}-l_{s+3}) + \cdots + q_{k-2}(l_{k-4}-l_{k-2}) \\ + q_{k-1}(l_{k-3}-l) + q_k(l_{k-2}-l)$$

(для нечетного $k-s$ выводится похожее выражение). Отсюда следует, что δ отрицательно, кроме случая $l_{k-2}=l$.

41. E F G H T U X Y Z V W B C D A P Q R J K L M I N O S ⊔.

42. Пусть $q_j = \mathtt{WT}(\mathtt{P}_j)$. Основная идея заключается в том, что на шагах C2–C6 *все* q_k становятся бо́льшими, чем начальное значение $q_{k-1}+q_k$, или равными ему.

43. Вызвав рекурсивную процедуру $mark(\mathtt{P}_1, 0)$, где $mark(\mathtt{P}, l)$ означает следующее:

$\mathtt{LEVEL(P)} \leftarrow l$;
если $\mathtt{LLINK(P)} \ne \Lambda$, то $mark(\mathtt{LLINK(P)}, l+1)$;
если $\mathtt{RLINK(P)} \ne \Lambda$, то $mark(\mathtt{RLINK(P)}, l+1)$.

44. Установите глобальные переменные $t \leftarrow 0$, $m \leftarrow 2n$ и вызовите рекурсивную подпрограмму $build(1)$, где $build(l)$ означает следующее.

Установить $j \leftarrow m$.
Если $\mathtt{LEVEL}(\mathtt{X}_t) = l$, то присвоить $\mathtt{LLINK}(\mathtt{X}_j) \leftarrow t$ и $t \leftarrow t+1$;
в противном случае присвоить $m \leftarrow m-1$, $\mathtt{LLINK}(\mathtt{X}_j) \leftarrow \mathtt{X}_m$ и $build(l+1)$.
Если $\mathtt{LEVEL}(\mathtt{X}_t) = l$, то присвоить $\mathtt{RLINK}(\mathtt{X}_j) \leftarrow t$ и $t \leftarrow t+1$;
в противном случае присвоить $m \leftarrow m-1$, $\mathtt{RLINK}(\mathtt{X}_j) \leftarrow \mathtt{X}_m$ и $build(l+1)$.

Переменная j локальна по отношению к построенной подпрограмме. (Это элегантное решение предложено Р. Е. Таржаном (R. E. Tarjan), *SICOMP* **6** (1977), 639.)
Внимание. Если числа $l_0, \ldots, l_n$ не соответствуют никакому бинарному дереву, алгоритм зациклится.

45. Представить рабочий массив $\mathtt{P}_0, \ldots, \mathtt{P}_t$ в виде списка с двойными связями, который также имеет связи со сбалансированным деревом (см. раздел 6.2.3). Если "попарно убывающие" веса равны $q_0, \ldots, q_t$ с q_j в корне дерева, можно перейти по дереву влево или вправо на основании значений q_j и q_{j+1}; двойные связи обеспечивают мгновенный доступ к q_{j+1}. (Поля RANK не являются необходимыми; при чередовании сохраняется симметричный порядок, а потому вносить изменения в двойные связи не нужно.)

Отдельные семейства весов, для которых задача может быть решена за время $O(n)$, были представлены Ху (Hu) и Моргенталером (Morgenthaler) в *Lecture Notes in Comp. Sci.* **1120** (1996), 234–243; однако неизвестно, достаточно ли времени $O(n)$ в общем случае.

46. См. *IEEE Trans.* **C-23** (1974), 268–271; кроме того, см. упр. 6.2.3–21.

47. См. Altenkamp and Mehlhorn, *JACM* **27** (1980), 412–427.

48. Не позволяйте сложному анализу случаев $N=3$ (Jonassen and Knuth, *J. Comp. Syst. Sci.* **16** (1978), 301–322) и $N=4$ (Baeza-Yates, *BIT* **29** (1989), 378–394) запугать вас! В этой области достигнут определенный прогресс (см. Louchard, Randrianarimanana, and Schott, *Theor. Comp. Sci.* **93** (1992), 201–225).

49. Этот вопрос был впервые исследован Дж. М. Робсоном (J. M. Robson) (*Australian Comp. J.* **11** (1979), 151–153), Б. Питтелем (B. Pittel) (*J. Math. Anal. Applic.* **103** (1984), 461–480) и Люком Девроем (Luc Devroye) (*JACM* **33** (1986), 489–498; *Acta Inf.* **24** (1987), 277–298), которые получили предельные формулы, выполняющиеся с вероятностью $\to 1$ при $n \to \infty$; см. H. M. Mahmoud, *Evolution of Random Search Trees* (Wiley, 1992), гла-

ва 2. Впоследствии Люком Девроем и Брюсом Ридом (Bruce Reed) (*SICOMP* **24** (1995), 1157–1162) был найден попахивающий шаманством результат — они доказали, что средняя высота равна $\alpha \ln n + O(\log\log n)$ с дисперсией $O(\log\log n)^2$, где

$$\alpha = 1/T(1/2e) \approx 4.31107\,04070\,01005\,03504\,70760\,96446\,89027\,83916-,$$

а $T(z) = \sum_{n=1}^{\infty} n^{n-1} z^n/n!$ представляет собой функцию дерева.

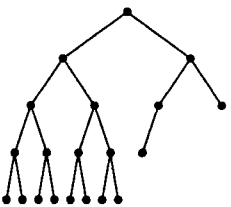
РАЗДЕЛ 6.2.3

1. При преобразованиях должен сохраняться симметричный порядок узлов; в противном случае получить бинарное дерево поиска невозможно.

2. B(S) = 0 только в том случае, когда S указывает на корень дерева (на шагах A3 и A4 значение S не изменялось) и все узлы от S до точки вставки сбалансированы.

3. Обозначим через ρ_h наибольшее возможное отношение числа несбалансированных узлов к общему количеству узлов в сбалансированном дереве высотой h. Тогда $\rho_1 = 0$, $\rho_2 = \frac{1}{2}$, $\rho_3 = \frac{1}{2}$. Докажем, что $\rho_h = (F_{h+1} - 1)/(F_{h+2} - 1)$. Пусть T_h — дерево, которое максимизирует значение ρ_h; тогда можно предположить, что его левое поддерево имеет высоту $h-1$, а правое — высоту $h-2$ (так как, если бы оба поддерева имели высоту $h-1$, интересующее нас отношение было бы меньше, чем ρ_{h-1}). Следовательно, это отношение для T_h не превышает $(\rho_{h-1}N_l + \rho_{h-2}N_r + 1)/(N_l + N_r + 1)$, где (N_l, N_r) — количество узлов в (левом, правом) поддереве. Приведенное выражение принимает максимальное значение при минимальных значениях (N_l, N_r). Значит, T_h представляет собой дерево Фибоначчи и согласно упр. 1.2.8–28 $\rho_h < \phi - 1$.

4. При $h = 7$ дерево



имеет бóльшую длину пути. [*Примечание.* В работе C. C. Foster, *Proc. ACM Nat. Conf.* **20** (1965), 197–198, приведена некорректная процедура построения N-узловых сбалансированных деревьев с максимальной длиной пути. Эдвард Логг (Edward Logg) обнаружил, что на рис. 3 у Фостера дан неоптимальный результат после 24 шагов (узел 22 может быть перемещен за узел 25).]

Дерево Фибоначчи порядка h, однако, минимизирует значение $(h+a)N -$ (длина внешнего пути(T)) по всем сбалансированным деревьям T высотой $h-1$ для любой неотрицательной константы a (это утверждение легко доказывается с помощью индукции по h). Длина его внешнего пути равна $\frac{3}{5}hF_{h-1} + \frac{4}{5}(h-1)F_h = (\phi/\sqrt{5})hF_{h+1} + O(F_{h+1}) = \Theta(h\phi^h)$. Следовательно, длина пути любого N-узлового сбалансированного дерева не превышает

$$\min_h(hN - \Theta(h\phi^h) + O(N)) \le N\log_\phi N - N\log_\phi\log_\phi N + O(N).$$

Более того, если N велико и $k = \lceil \lg N \rceil$, $h = \lfloor k/\lg\phi - \log_\phi k \rfloor = \log_\phi N - \log_\phi\log_\phi N + O(1)$, можно построить сбалансированное дерево с длиной пути $hN + O(N)$ следующим образом: запишем $N + 1 = F_h + F_{h-1} + \cdots + F_{k+1} + N' = F_{h+2} - F_{k+2} + N'$ и построим бинарное дерево с N' узлами; затем последовательно объединим его с деревьями Фибоначчи порядков k, $k+1$, ..., $h-1$ [см. R. Klein and D. Wood, *Theoretical Comp. Sci.* **72** (1990), 251–264].

5. Это утверждение может быть доказано по индукции. Если T_N обозначает построенное дерево, имеем

$$T_N = \begin{cases} \bigcirc\!\!\!

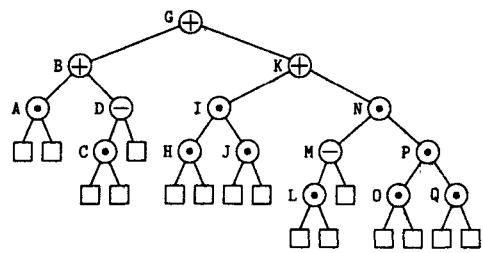
12. Максимально возможное время достигается при вставке во второй внешний узел (12); $C = 4$, $C1 = 3$, $D = 3$, $A = C2 = F = G1 = H1 = U1 = 1$, и общее время равно $132u$. Минимум достигается при вставке в третий от конца внешний узел (13); $C = 2$, $C1 = C2 = 1$, $D = 2$, и общее время равно $61u$. (Соответствующие параметры программы 6.2.2T равны $74u$ и $26u$.)

13. При изменениях дерева должны обновляться только $O(\log N)$ значений RANK; "упрощенная" система может потребовать большего количества изменений.

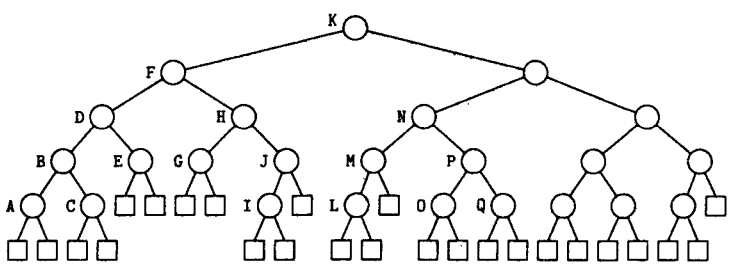
14. Да (хотя типичные операции над списками весьма неслучайны и вероятность появления вырожденных деревьев достаточно высока).

15. Воспользуйтесь алгоритмом 6.2.2T с установкой $m \leftarrow 0$ на шаге T1 и $m \leftarrow m + \texttt{RANK(P)}$ при $K \geq \texttt{KEY(P)}$ на шаге T2.

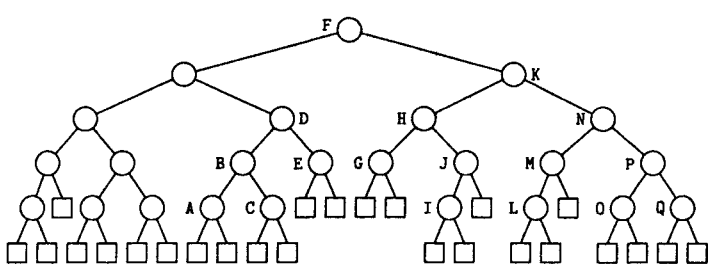
16. Удалим E; выполним ребалансировку (случай 3) в D. Удалим G; заменим F на G; выполним ребалансировку (случай 2) в H; откорректируем фактор сбалансированности в K.



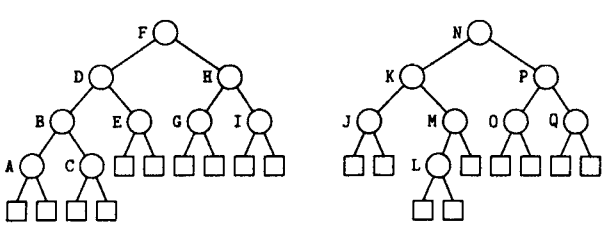
17. (a)



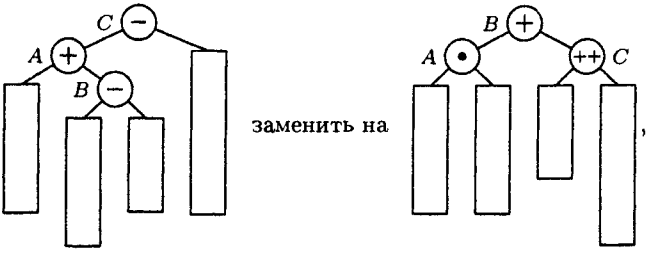
(b)



18.

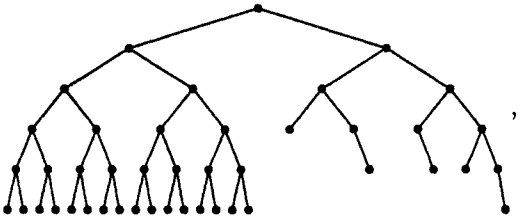


19. (Решение Кларка Крейна (Clark Crane).) Имеется один случай, который не может быть сведен к однократному или двукратному повороту в корне. В этом случае следует



а затем избавиться от несбалансированности с помощью однократного или двукратного поворота в C.

20. Очень сложно вставить новый узел в крайнюю слева позицию показанного дерева:



однако К.-Ю. Ряйхя (K.-J. Räihä) и С. Г. Цвебен (S. H. Zweben) разработали алгоритм вставки, который требует $O(\log N)$ шагов [*CACM* **22** (1979), 508–512].

21. Алгоритм А выполняет задание за $N \log N$ шагов (см. упр. 5); описанный далее алгоритм создает такое же дерево за $O(N)$ шагов при помощи интересной итеративной реализации рекурсивного метода. Мы используем три вспомогательных списка:

$\mathtt{D}_1, \ldots, \mathtt{D}_l$ (бинарный счетчик, который управляет рекурсией);

$\mathtt{J}_1, \ldots, \mathtt{J}_l$ (список указателей на стыковочные узлы);

$\mathtt{T}_1, \ldots, \mathtt{T}_l$ (список указателей на деревья).

Здесь $l = \lceil \lg(N+1) \rceil$. Для удобства алгоритм также устанавливает $\mathtt{D}_0 \leftarrow 1$, $\mathtt{J}_0 \leftarrow \mathtt{J}_{l+1} \leftarrow \Lambda$.

G1. [Инициализация.] Установить $l \leftarrow 0$, $\mathtt{J}_0 \leftarrow \mathtt{J}_1 \leftarrow \Lambda$, $\mathtt{D}_0 \leftarrow 1$.

G2. [Получить следующий элемент.] Пусть P указывает на следующий входящий узел (для его получения может использоваться другая программа). Если новых узлов больше нет, переходим к шагу G5. В противном случае установить $k \leftarrow 1$, $\mathtt{Q} \leftarrow \Lambda$ и заменить $\mathtt{P} \leftrightarrow \mathtt{J}_1$.

G3. [Продолжение.] Если $k > l$ (или, что то же самое, $\mathtt{P} = \Lambda$), установить $l \leftarrow l + 1$, $\mathtt{D}_k \leftarrow 1$, $\mathtt{T}_k \leftarrow \mathtt{Q}$, $\mathtt{J}_{k+1} \leftarrow \Lambda$ и вернуться к шагу G2. В противном случае установить $\mathtt{D}_k \leftarrow 1 - \mathtt{D}_k$, заменить $\mathtt{Q} \leftrightarrow \mathtt{T}_k$, $\mathtt{P} \leftrightarrow \mathtt{J}_{k+1}$ и увеличить k на 1. Если теперь $\mathtt{D}_{k-1} = 0$, повторить этот шаг.

G4. [Конкатенация.] Установить $\mathtt{LLINK(P)} \leftarrow \mathtt{T}_k$, $\mathtt{RLINK(P)} \leftarrow \mathtt{Q}$, $\mathtt{B(P)} \leftarrow 0$, $\mathtt{T}_k \leftarrow \mathtt{P}$ и вернуться к шагу G2.

G5. [Завершение.] Установить $\mathtt{LLINK(J}_k\mathtt{)} \leftarrow \mathtt{T}_k$, $\mathtt{RLINK(J}_k\mathtt{)} \leftarrow \mathtt{J}_{k-1}$, $\mathtt{B(J}_k\mathtt{)} \leftarrow 1 - \mathtt{D}_{k-1}$ для $1 \le k \le l$. Алгоритм завершен ($\mathtt{J}_l$ указывает на корень искомого дерева). ▮

Шаг G3 выполняется $2N - \nu(N)$ раз, где $\nu(N)$ — число единиц в двоичном представлении числа N.

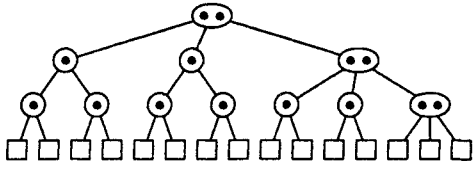
22. Высота взвешенно-сбалансированного дерева с N внутренними узлами всегда лежит между $\lg(N+1)$ и $2\lg(N+1)$. Для получения верхней границы заметьте, что более тяжелое поддерево корня имеет не более $(N+1)/\sqrt{2}$ внешних узлов.

23. (a) Постройте дерево, правое поддерево которого представляет собой полное бинарное дерево с $2^n - 1$ узлами, а левое — дерево Фибоначчи с $F_{n+1} - 1$ узлами. (b) Постройте взвешенно-сбалансированное дерево, правое поддерево которого имеет высоту порядка $2\lg N$, а левое — порядка $\lg N$ (см. упр. 22).

24. Рассмотрим наименьшее дерево, удовлетворяющее условию, но не являющееся идеально сбалансированным. Тогда его левое и правое поддеревья идеально сбалансированы и соответственно количество их внешних узлов составляет 2^l и 2^r, где $l \neq r$, что противоречит заданному условию.

25. После вставки узла в нижнюю часть дерева мы движемся вверх, проверяя весовой баланс в каждом узле на пути поиска. Предположим, что в узле A в (1) имеется несбалансированность, а новый узел вставлен в правое поддерево, где B и его поддеревья взвешенно-сбалансированы. Тогда после однократного поворота баланс восстановится, если $(|\alpha| + |\beta|)/|\gamma| > \sqrt{2} + 1$, где $|x|$ означает число внешних узлов в дереве x. Однако можно показать, что в этом случае достаточно двукратного поворота [см. *SICOMP* **2** (1973), 33–43].

27. Иногда в узлах с двумя ключами необходимо выполнить два сравнения. Наихудший случай встречается в деревьях, подобных изображенному, когда в некоторых ситуациях требуется $2\lg(N+2) - 2$ сравнений.



29. Частичное решение, принадлежащее Э. Яо (A. Yao), таково: для $N \geq 6$ ключей нижний уровень будет содержать в среднем $\frac{2}{7}(N+1)$ узлов с одним ключом и $\frac{1}{7}(N+1)$ узлов — с двумя ключами. Общее среднее количество узлов при больших N лежит между $0.70N$ и $0.79N$. [*Acta Informatica* **9** (1978), 159–170.]

30. Для случая наилучшего подходящего упорядочите записи по размерам по произвольному правилу связывания областей с одним и тем же размером (см. упр. 2.5–9). В случае первого подходящего упорядочите записи по адресам с дополнительным полем в каждом узле, содержащем размер наибольшей области в поддереве, для которого данный узел является корнем. Эти поля могут обновляться при вставках и удалениях. (Впрочем, хотя время работы и оказывается равным $O(\log n)$, вероятно, на практике метод блужданий из упр. 2.5–6 будет более эффективным. Однако без ROVER память может распределяться еще лучше, так как обычно "на всякий пожарный" поддерживается свободной область памяти большого объема.)

В работе R. P. Brent, *ACM Trans. Prog. Languages and Systems* **11** (1989), 388–403, можно ознакомиться с усовершенствованием описанного метода.

31. Используйте почти сбалансированное дерево с дополнительными связями вверх для крайней слева части и стек отложенных корректировок фактора сбалансированности вдоль этого пути (для каждой вставки требуется ограниченное число таких корректировок).

Эта задача может быть обобщена на случай использования $O(\log m)$ шагов для поиска, вставки и/или удаления элементов, которые находятся в m шагах от данного "указателя"; в

качестве такого указателя может выступать любой узел, расположение которого известно. [См. S. Huddleston and K. Mehlhorn, *Acta Inf.* **17** (1982), 157–184.]

32. При каждом вращении вправо увеличивается один из r_k, не изменяя остальных, откуда $r_k \le r'_k$. Чтобы показать, что этого достаточно, предположим, что $r_j = r'_j$ для $1 \le j < k$, но $r_k < r'_k$. Тогда существует вращение вправо, при котором r_k увеличивается до значения $\le r'_k$, потому что числа $r_1 r_2 \ldots r_n$ удовлетворяют условию упр. 2.3.3–19, (a).

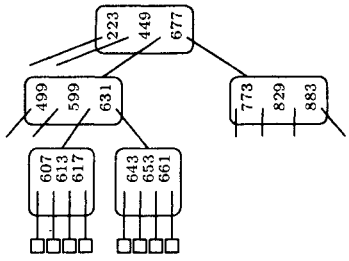
Примечание. Этот частичный порядок, впервые введенный в 1951 году Д. Тамари (D. Tamari), имеет много интересных свойств. Любые два дерева имеют наибольшую нижнюю границу $T \wedge T'$, определяемую размерами правых поддеревьев $\min(r_1, r'_1)\min(r_2, r'_2)\ldots\min(r_n, r'_n)$, так же, как и наименьшая верхняя граница $T \vee T'$ определяется размерами левых поддеревьев $\min(l_1, l'_1)\min(l_2, l'_2)\ldots\min(l_n, l'_n)$. Размеры левых поддеревьев, конечно, на единицу меньше, чем поля RANK в алгоритмах B и C. За дополнительной информацией обратитесь к работам H. Friedman and D. Tamari, *J. Combinatorial Theory* **2** (1967), 215–242, **4** (1968), 201; C. Greene, *Europ. J. Combinatorics* **9** (1988), 225–240; D. D. Sleator, R. E. Tarjan, and W. P. Thurston, *J. Amer. Math. Soc.* **1** (1988), 647–681; J. M. Pallo, *Theoretical Informatics and Applic.* **27** (1993), 341–348; M. K. Bennett and G. Birkhoff, *Algebra Universalis* **32** (1994), 115–144; P. H. Edelman and V. Reiner, *Mathematika* **43** (1996), 127–154.

33. Во-первых, можно свести объем памяти к одному биту A(P) в каждом узле P, такому, что B(P) = A(RLINK(P)) − A(LLINK(P)), когда LLINK(P) и RLINK(P) ненулевые; в противном случае B(P) уже известно. Во-вторых, можно положить, что A(P) = 0, когда LLINK(P) и RLINK(P) нулевые. Тогда A(P) может быть удален во всех других узлах после обмена LLINK(P) с RLINK(P) при A(P) = 1; сравнением KEY(P) с KEY(LLINK(P)) или KEY(RLINK(P)) определяется A(P).

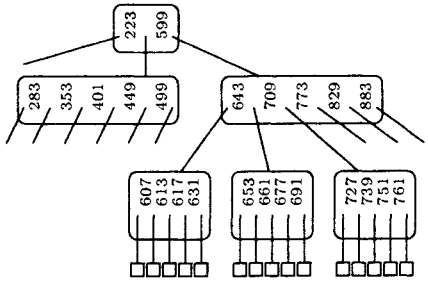
Естественно, на машинах, указатели которых всегда четны, каждый узел имеет два неиспользуемых бита. Чтобы получить дополнительную экономию памяти, следует поступить, как в упр. 2.3.1–37.

РАЗДЕЛ 6.2.4

1. Разделяются два узла:



2. Измененные узлы:



(Конечно, B^*-дерево могло бы и не иметь узлов с тремя ключами, хотя на рис. 30 они показаны.)

3. (a) $1 + 2 \cdot 50 + 2 \cdot 51 \cdot 50 + 2 \cdot 51 \cdot 51 \cdot 50 = 2 \cdot 51^3 - 1 = 265301$.
(b) $1 + 2 \cdot 50 + (2 \cdot 51 \cdot 100 - 100) + ((2 \cdot 51 \cdot 101 - 100) \cdot 100 - 100) = 101^3 = 1030301$.
(c) $1 + 2 \cdot 66 + (2 \cdot 67 \cdot 66 + 2) + (2 \cdot 67 \cdot 67 \cdot 66 + 2 \cdot 67) = 601661$. (Меньше, чем (b)!)

4. Перед разделением некорневого узла убедитесь, что он имеет два заполненных соседа. Затем разделите эти три узла на четыре. Корень должен разделяться только тогда, когда в нем содержится больше $3\lfloor(3m-3)/4\rfloor$ ключей.

5. Интерпретация 1, попытка максимизировать сформулированную задачу дает 450. (Наихудшая ситуация возникает при наличии 1 005 символов и передаваемого родительскому узлу ключа длиной 50: 445 символов + указатель + 50-символьный ключ + указатель + 50-символьный ключ + указатель + 445 символов.)

Интерпретация 2, попытка уравнять количество ключей после разделения для поддержания высокого ветвления: 155 (15 коротких ключей, за которыми следуют 16 длинных).

См. E. M. McCreight, *CACM* **20** (1977), 670–674.

6. Если удаляемый ключ находится не на уровне $l-1$, замените его ключом-наследником и удалите последний. Для удаления ключа на уровне $l-1$ достаточно просто стереть его; если при этом узел окажется слишком пустым, обратитесь к его правому (или левому) соседу и выполните "вливание", т. е. переместите ключи в узел из соседа таким образом, чтобы в обоих узлах содержалось примерно одинаковое количество данных. Такая операция невозможна только при малой заполненности соседа; в этом же случае можно объединить два узла в один (вместе с одним ключом из родительского узла). Такое объединение может, в свою очередь, вызвать необходимость вливания на уровне родительского узла и т. д. При наличии ключей переменной длины, как в упр. 5, может потребоваться разделение родительского узла (когда один из его ключей становится длиннее).

8. Имея дерево $\mathcal{T}$ с N внутренними узлами, обозначим через $a_k^{(j)}$ внешние узлы, требующие k обращений, родительские узлы которых относятся к страницам, содержащим j ключей. Пусть также $A^{(j)}(z)$ является соответствующей производящей функцией. Тогда $A^{(1)}(1) + \cdots + A^{(M)}(1) = N + 1$. (Заметим, что $a_k^{(j)}$ кратно $j+1$ при $1 \le j < M$.) Следующая случайная вставка приводит к появлению $N+1$ равновероятного дерева, производящие функции которых получаются путем уменьшения некоторых коэффициентов $a_k^{(j)}$ на $j+1$ и прибавления $j+2$ к $a_k^{(j+1)}$ (или, если $j = M$, уменьшения некоторых $a_k^{(M)}$ на 1 и прибавления 2 к $a_{k+1}^{(1)}$). Теперь $B_N^{(j)}(z)$ равно $(N+1)^{-1}$, умноженному на сумму производящих функций $A^{(j)}(z)$ для $\mathcal{T}$, умноженных на вероятность появления $\mathcal{T}$ по всем деревьям $\mathcal{T}$. Отсюда следуют такие сформулированные рекуррентные соотношения:

$$\begin{aligned}(B_N^{(1)}(z), \ldots, B_N^{(M)}(z))^T &= (I + (N+1)^{-1}W(z))(B_{N-1}^{(1)}(z), \ldots, B_{N-1}^{(M)}(z))^T \\ &= \cdots = g_N(W(z))(0, \ldots, 0, 1)^T,\end{aligned}$$

где

$$g_n(x) = \left(1 + \frac{x}{n+1}\right) \cdots \left(1 + \frac{x}{2}\right) = \frac{1}{x+1}\binom{x+n+1}{n+1}.$$

Значит, $C'_N = (1, \ldots, 1)(B_N^{(1)\prime}(1), \ldots, B_N^{(M)\prime}(1))^T = 2B_{N-1}^{(M)}(1)/(N+1) + C'_{N-1} = 2f_N(W)_{MM}$, где $f_n(x) = g_{n-1}(x)/(n+1) + \cdots + g_0(x)/2 = (g_n(x)-1)/x$ и $W = W(1)$. (Индекс MM означает нижний левый угол матрицы.) Теперь $W = S^{-1}\operatorname{diag}(\lambda_1, \ldots, \lambda_M)S$ для некоторой матрицы S, где $\operatorname{diag}(\lambda_1, \ldots, \lambda_M)$ обозначает диагональную матрицу, элементами которой являются корни полинома $\chi(\lambda) = (\lambda+2)\ldots(\lambda+M+1) - (M+1)!$. (Все корни различны, поскольку $\chi(\lambda) = \chi'(\lambda) = 0$ приводит к $1/(\lambda+2) + \cdots + 1/(\lambda+M+1) = 0$; это может выполняться только тогда, когда λ действительно и $-M-1 < \lambda < -2$, откуда вытекает, что $|\lambda+2|\ldots|\lambda+M+1| <$

$(M+1)!$, а это противоречит начальному условию.) Если $p(x)$ — некоторый полином, то $p(W) = p(S^{-1}\operatorname{diag}(\lambda_1,\ldots,\lambda_M)S) = S^{-1}\operatorname{diag}(p(\lambda_1),\ldots,p(\lambda_M))S$; следовательно, правый нижний элемент $p(W)$ имеет вид $c_1p(\lambda_1)+\cdots+c_Mp(\lambda_M)$, где некоторые константы $c_1,\ldots,c_M$ не зависят от p. Эти константы можно вычислить, положив $p(\lambda) = \chi(\lambda)/(\lambda-\lambda_j)$; так как $(W^k)_{MM} = (-2)^k$ при $0 \le k \le M-1$, имеем $p(W)_{MM} = p(-2) = (M+1)!/(\lambda_j+2) = c_jp(\lambda_j) = c_j\chi'(\lambda_j) = c_j(M+1)!\,(1/(\lambda_j+2)+\cdots+1/(\lambda_j+M+1))$. Значит, $c_j = (\lambda_j+2)^{-1}(1/(\lambda_j+2)+\cdots+1/(\lambda_j+M+1))^{-1}$. Это дает "точную" формулу $C'_N = \sum_{j=1}^{M} 2c_jf_N(\lambda_j)$; остается только исследовать корни λ_j. Заметим, что $|\lambda_j+M+1| \le M+1$ для всех j, иначе мы бы имели $|\lambda_j+2|\ldots|\lambda_j+M+1| > (M+1)!$. Взяв $\lambda_1 = 0$, убеждаемся, что $\Re(\lambda_j) < 0$ для $2 \le j \le M$. Согласно 1.2.5–(15) $g_n(x) \sim (n+1)^x/\Gamma(x+2)$ при $n \to \infty$; следовательно, $g_n(\lambda_j) \to 0$ для $2 \le j \le M$. Таким образом, $C'_N = 2c_1f_N(0) + O(1) = H_N/(H_{M+1}-1) + O(1)$.

Примечание. Приведенный анализ применим также к алгоритму простой сортировки, вкратце обсуждавшемуся в разделе 5.2.2. Вычисления могут быть легко дополнены для того, чтобы показать, что $B_N^{(j)}(1) \sim (H_{M+1}-1)^{-1}/(j+2)$ при $1 \le j < M$ и $B_N^{(M)}(1) \sim (H_{M+1}-1)^{-1}/2$. Поэтому общее количество внутренних узлов в незаполненных страницах приблизительно равно

$$\left(\frac{1}{3\times 2}+\frac{2}{4\times 3}+\cdots+\frac{M-1}{(M+1)\times M}\right)\frac{N}{H_{M+1}-1} = \left(1-\frac{M}{(M+1)(H_{M+1}-1)}\right)N,$$

а общее количество использованных страниц примерно равно

$$\left(\frac{1}{3\times 2}+\frac{1}{4\times 3}+\cdots+\frac{1}{(M+1)\times M}+\frac{1}{M+1}\right)\frac{N}{H_{M+1}-1} = \frac{N}{2(H_{M+1}-1)},$$

откуда следует, что асимптотическое использование памяти равно $2(H_{M+1}-1)/M$.

Этот анализ был развит в работе Махмуда (Mahmoud) и Питтеля (Pittel) [*J. Algorithms* **10** (1989), 52–75], которые открыли, что дисперсия количества использованной памяти подвергается неожиданному фазовому переходу: при $M \le 25$ дисперсия равна $\Theta(N)$, но при $M \ge 26$ она асимптотически равна $f(N)N^{1+2\alpha}$, где $f(e^{\pi/\beta}N) = f(N)$, если $-\frac{1}{2}+\alpha+\beta i$ и $-\frac{1}{2}+\alpha-\beta i$ являются ненулевыми корнями λ_j с наибольшей действительной частью.

Высота таких деревьев проанализирована в работах L. Devroye, *Random Structures and Algorithms* **1** (1990), 191–203, и B. Pittel, *Random Structures and Algorithms* **5** (1994), 337–347.

9. Да; например, мы могли бы заменить каждое K_i в (1) на i плюс число ключей в поддеревьях $\mathrm{P}_0,\ldots,\mathrm{P}_{i-1}$. Соответствующим образом изменяются алгоритмы поиска, вставки и удаления.

10. Краткий набросок: расширим страничную схему так, чтобы исключительный доступ к буферам предоставлялся только одному пользователю одновременно. Алгоритмы поиска, вставки и удаления должны быть тщательно модифицированы, чтобы такой исключительный доступ предоставлялся только на ограниченное время, только при крайней необходимости и таким образом, чтобы не возникало клинчей. За подробностями обратитесь к работам B. Samadi, *Inf. Proc. Letters* **5** (1976), 107–112; R. Bayer and M. Schkolnick, *Acta Inf.* **9** (1977), 1–21; Y. Sagiv, *J. Comp. Syst. Sci.* **33** (1986), 275–296.

РАЗДЕЛ 6.3

1. Отблески. (На самом деле перед нами непереводимая игра слов, основанная на близости слов *tree* и *trie*. В связи с этим ниже приводится текст оригинала упражнения и ответа к нему. — *Прим. перев.*

If a tree has leaves, what does a trie have?

Lieves (the plural of "lief").)

2. Выполните алгоритм T, используя в качестве аргумента новый ключ; поиск закончится неудачно на шаге T3 или T4. Если последний шаг — T3, просто поместите K в элемент таблицы NODE(P) с номером k и завершите работу алгоритма. В противном случае поместите в эту позицию адрес нового узла Q ⇐ AVAIL, содержащего пустые ссылки, а затем установите P ← Q. Теперь присвойте k и k' соответственно следующие символы K и X: если $k \neq k'$, сохраните K в позиции k узла NODE(P), а X — в позиции k'; однако, если $k = k'$, поместите в позицию k указатель на новый узел Q ⇐ AVAIL и присвойте P ← Q. Повторяйте этот процесс до тех пор, пока не получится $k \neq k'$ (считается, что ни один из ключей не служит началом другому).

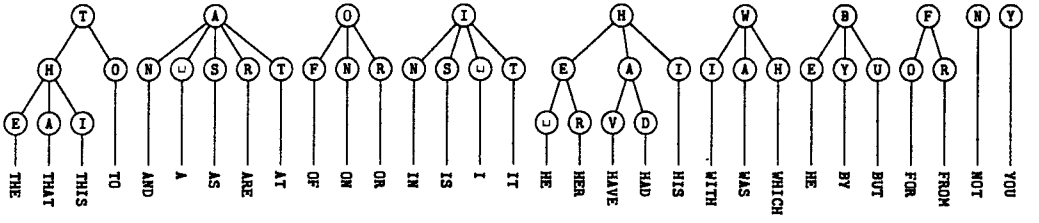
3. Заменим ключ пустой ссылкой в том узле, в котором он находился. Если теперь данный узел стал "бесполезным" в силу того, что все его элементы, кроме одного, представляющего ключ X, пусты, удалим этот узел и заменим соответствующий указатель в родительском узле указателем X. Если "бесполезным" становится родительский узел, удалим его описанным способом.

4. Успешный поиск будет проходить так же, как и в случае несжатой таблицы, однако для неудачного поиска в сжатой таблице может потребоваться несколько дополнительных итераций. Например, такой аргумент поиска, как TRASH, заставит программу произвести *шесть* итераций (что явно больше пяти!); этот случай является наихудшим. Необходимо также проверить невозможность зацикливания на пустых позициях. (Этот замечательный пример упаковки таблицы в 49 элементов предложен Дж. С. Фишберном (J. Scot Fishburn), который также доказал, что 48 позиций для упаковки таблицы недостаточно.)

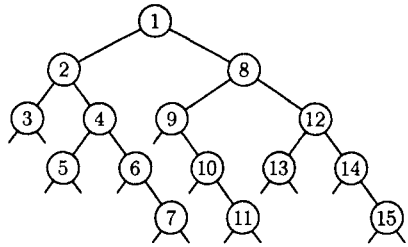
Более медленный, но более многосторонний путь экономии памяти для луча был предложен Куртом Мали (Kurt Maly), *CACM* **19** (1976), 409–415.

В общем, если нужно сжать n разреженных таблиц, содержащих соответственно x_1, ..., x_n ненулевых элементов, метод "первый подходящий", который смещает j-ю таблицу на минимальную величину r_j с целью избежания конфликтов с ранее размещенными таблицами даст $r_j \leq (x_1 + \cdots + x_{j-1})x_j$, поскольку каждый предшествующий ненулевой элемент может блокировать не более x_j смещений. Такая оценка наихудшего случая дает $r_j \leq 93$ для табл. 1, гарантируя, что любые 12 таблиц длиной 30, содержащие соответственно 10, 5, 4, 3, 3, 3, 3, 3, 2, 2, 2, 2 ненулевых элементов, могут быть упакованы в $93 + 30$ последовательных позиций безотносительно к двоичному представлению ненулевых элементов. Дальнейшее усовершенствование этого метода было проведено в работе R. E. Tarjan and A. C. Yao, *CACM* **22** (1979), 606–611. Динамическая реализация сжатых лучей, предложенная Ф. М. Лиангом (F. M. Liang), использована в таблицах переносов в издательской системе TEX (см. D. E. Knuth, *CACM* **29** (1986), 471–478; *Literate Programming* (1991), 206–233).

5. В каждом семействе сначала проверяется наиболее вероятный исход; для этого буквы располагаются слева направо в порядке уменьшения вероятности. Оптимальность такого упорядочения может быть доказана аналогично теореме 6.1S. [См. *CACM* **12** (1969), 72–76.]



6.



7. Например, 8, 4, 1, 2, 3, 5, 6, 7, 12, 9, 10, 11, 13, 14, 15. (Независимо от того, какая последовательность используется, ни левое, ни правое поддерево не может содержать больше двух узлов на уровне 4.) Даже полученное нами "наихудшее" дерево принадлежит к числу четырех наилучших возможных деревьев, так что, как видите, деревья цифрового поиска не слишком чувствительны к порядку вставки.

8. Да. В поле KEY теперь содержится только усеченный ключ; левые биты, значения которых определяются позицией узла, удалены. Аналогичная модификация применима и к алгоритму T.

9.

START	LDX	K	1	*D1. Инициализация.*	(rX $\equiv K$)
	LD1	ROOT	1	P $\leftarrow$ ROOT.	(rI1 $\equiv$ P)
	JMP	2F	1		
4H	LD2	0,1(RLINK)	$C2$	*D4. Перемещение вправо.*	Q $\leftarrow$ RLINK(P).
	J2Z	5F	$C2$	Переход к шагу D5 при Q $= \Lambda$.	
1H	ENT1	0,2	$C-1$	P $\leftarrow$ Q.	
2H	CMPX	1,1	C	*D2. Сравнение.*	
	JE	SUCCESS	C	Выход при $K =$ KEY(P).	
	SLB	1	$C-S$	Смещение K влево на один бит.	
	JAO	4B	$C-S$	Переход к шагу D4, если выделенный бит равен 1.	
	LD2	0,1(LLINK)	$C1$	*D3. Перемещение влево.*	Q $\leftarrow$ LLINK(P).
	J2NZ	1B	$C1$	Переход к шагу D2 с P $\leftarrow$ Q, если Q $\neq \Lambda$.	
5H	Продолжение то же, что и в программе 6.2.2T, с обменом местами rA и rX. ▮				

Время работы фазы поиска этой программы равно $(10C - 3S + 4)u$, где $C - S$ представляет собой число проверок битов. Для случайных данных приближенное среднее время работы таково.

	Успешно	Неудачно
Программа 6.2.2T	$15 \ \ln N - 12.34$	$15 \ \ln N - 2.34$
Данная программа	$14.4 \ln N - \ 6.17$	$14.4 \ln N + 1.26$

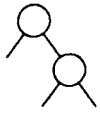
(Следовательно, при не очень больших N программа 6.2.2T работает несколько быстрее.)

10. Обозначим через $\oplus$ операцию "исключающее или" над n-битовыми числами, и пусть $f(x) = n - \lceil \lg(x+1) \rceil$ — число левых ненулевых битов x. Вот одно из решений: (b) если поиск с помощью алгоритма T заканчивается неудачно на шаге T3, то k на единицу меньше количества проверенных битов; в противном случае, если поиск заканчивается на шаге T4, $k = f(K \oplus X)$. (a, c) Выполним обычный поиск, но при этом будем хранить x — минимальное значение $K \oplus$ KEY(P) по всем KEY(P), сравнивавшимся с K в процессе поиска. Тогда $k = f(x)$. (Докажите, что наибольшее число битов, совпадающих с K, имеют ключи, сравниваемые с аргументом. В случае (a) максимум k достигается либо для наибольшего ключа $\leq K$, либо для наименьшего ключа $> K$.)

11. Нет; удаление узла только с одним пустым поддеревом приведет к потере одного бита в ключах непустого поддерева. Для удаления узла следует заменить его одним из

его *терминальных* наследников, например, по возможности двигаясь при поиске только вправо.

12. Вставим три случайных числа — α, β, γ из диапазона $[0,1]$ — в изначально пустое дерево; затем удалим α с вероятностью p, β с вероятностью q, γ с вероятностью r с помощью предложенного в предыдущем упражнении алгоритма. Дерево



получается с вероятностью $\frac{1}{4}p + \frac{1}{2}q + \frac{1}{2}r$, а это равно $\frac{1}{2}$ только при $p = 0$.

13. Добавить к каждому узлу поле KEY и сравнивать K с этим ключом перед проверкой элемента вектора на шаге T2. Табл. 1 должна быть изменена следующим образом: узлы (1), ..., (12) должны содержать ключи THE, AND, BE, FOR, HIS, IN, OF, TO, WITH, HAVE, HE, THAT соответственно (если они вставляются в порядке уменьшения частоты), и эти ключи должны быть удалены из своих прежних позиций. (Соответствующая программа будет медленнее и сложнее программы T. Более прямолинейное M-арное обобщение алгоритма D привело бы к созданию дерева с N узлами с одним ключом и M ссылками в каждом узле.)

14. Если $j \le n$, то имеется только одно такое место, а именно — KEY(P). Однако при $j > n$ множество всех появлений находится посредством обхода поддерева узла P: если имеется r появлений K в TEXT, то поддерево содержит $r - 1$ узел (включая узел P) и, таким образом, имеет r полей ссылок с TAG = 1. Эти поля ссылок указывают на все узлы, ссылающиеся на позиции в TEXT, которые содержат K (в повторных проверках TEXT нет необходимости).

15. Начните построение дерева с установки KEY(HEAD) равной первой ссылке на TEXT, а также LLINK(HEAD) $\leftarrow$ HEAD, LTAG(HEAD) $\leftarrow$ 1. Дальнейшие ссылки на TEXT могут быть вставлены в дерево с использованием следующего алгоритма.

Установить K равным значению нового ключа, который следует вставить (это первая ссылка на массив TEXT, которую делает алгоритм). Выполните алгоритм P. Он должен закончиться неудачно, поскольку ни один ключ не может быть префиксом другого ключа. (На шаге P6 осуществляется вторая (и последняя) ссылка на массив TEXT; больше ссылки на TEXT делаться не будут). Предположим теперь, что ключ, найденный на шаге P6, согласуется с аргументом K по первым l бит, но отличается, начиная с позиции $l + 1$ (в которой K имеет цифру b, а ключ соответственно цифру $1-b$). (Хотя при поиске с помощью алгоритма P ключ j может стать намного больше, чем l, можно доказать, что описанная здесь процедура найдет наилучшее совпадение с K среди *всех* имеющихся ключей. Таким образом, во *всех* ключах текста, которые совпадают с K по первым l бит, в качестве $(l+1)$-го бита содержится $1 - b$.) Теперь повторим выполнение алгоритма P, в котором K заменено его ведущими l бит (т. е. $n \leftarrow l$). На этот раз поиск будет успешным, поэтому шаг P6 выполняться не будет. Установите R $\Leftarrow$ AVAIL, KEY(R) $\leftarrow$ положение нового ключа в TEXT. Если LLINK(Q) = P, установите LLINK(Q) $\leftarrow$ R, $t \leftarrow$ LTAG(Q), LTAG(Q) $\leftarrow$ 0; в противном случае установите RLINK(Q) $\leftarrow$ R, $t \leftarrow$ RTAG(Q), RTAG(Q) $\leftarrow$ 0. Если $b = 0$, установите LTAG(R) $\leftarrow$ 1, LLINK(R) $\leftarrow$ R, RTAG(R) $\leftarrow t$, RLINK(R) $\leftarrow$ P; иначе установите RTAG(R) $\leftarrow$ 1, RLINK(R) $\leftarrow$ R, LTAG(R) $\leftarrow t$, LLINK(R) $\leftarrow$ P. Если $t = 1$, установите SKIP(R) $\leftarrow 1 + l - j$; в противном случае установите SKIP(R) $\leftarrow 1 + l - j +$ SKIP(P) и SKIP(P) $\leftarrow j - l - 1$.

16. Структура дерева требует, чтобы в каждый узел входила одна идущая снизу пунктирная линия, которая начинается в той части дерева, где соответствующий ключ впервые отличается от всех остальных. Если такой части дерева нет, алгоритмы перестают работать. Мы можем просто убрать ключи, служащие началом других ключей, но тогда алгоритм из упр. 14 не получит достаточного количества данных для поиска *всех* вхождений аргумента.

17. Если положить $a_0 = a_1 = 0$, то

$$x_n = a_n + \sum_{k\ge 2}\binom{n}{k}(-1)^k\hat a_k/(m^{k-1}-1) = \sum_{k\ge 2}\binom{n}{k}(-1)^k\hat a_k m^{k-1}/(m^{k-1}-1).$$

18. Для решения (4) необходимо преобразование $a_n = [n>1]$, а именно — $\hat a_n = [n=0] - 1 + n$; следовательно, для $N \ge 2$ получим $A_N = 1 - U_N + V_N$, где $U_N = K(N,0,M)$ и $V_N = K(N,1,M)$ (в обозначениях из упр. 19). Аналогично для решения (5) нужно взять $a_n = n - [n=1] = \hat a_n$ и получить $C_N = N + V_N$ для $N \ge 2$.

19. При $s = 1$ имеем $V_n = K(n,1,m) = n((\ln n+\gamma)/\ln m - \frac12 - \delta_0(n-1)) + O(1)$, а при $s \ge 2$ имеем $K(n,s,m) = (-1)^s n(1/\ln m + \delta_{s-1}(n-s))/s(s-1) + O(1)$, где

$$\delta_s(n) = \frac{2}{\ln m}\sum_{k\ge 1}\Re(\Gamma(s - 2\pi ik/\ln m)\exp(2\pi ik\log_m n))$$

представляет собой периодическую функцию от $\log n$. [В этом выводе использовался тот факт, что

$$K(n+s,s,m)/(-1)^s\binom{n+s}{s} = \frac{n^{-s+1}}{2\pi i}\int_{1/2-i\infty}^{1/2+i\infty}\frac{\Gamma(z)n^{s-1-z}\,dz}{m^{s-1-z}-1} + O(n^{-s}).$$

Для малых m и s весьма малым является δ (см. упр. 5.2.2–46). Обратите внимание на то, что $\delta_s(n-a) = \delta_s(n) + O(n^{-1})$ при фиксированном a.]

20. Для случая (a) положим $a_n = [n>s] = 1 - \sum_{k=0}^s[n=k]$; для (b) — $a_n = n - \sum_{k=0}^s k[n=k]$; в случае (c) требуется найти решение рекуррентного соотношения

$$y_n = \begin{cases} m^{1-n}\sum_k\binom{n}{k}(m-1)^{n-k}y_k & \text{при } n > s,\\ \binom{n+1}{2} & \text{при } n \le s.\end{cases}$$

Подставив $x_n = y_n - n$, получим рекуррентное соотношение рассмотренной в упр. 17 формы, где

$$a_n = (1-M^{-1})\sum_{k=0}^s\binom{k}{2}[n=k].$$

Таким образом, используя обозначения из предыдущих упражнений, получаем следующие ответы: (a) $1 - K(N,0,M) + K(N,1,M) - \cdots + (-1)^{s-1}K(N,s,M) = N/(s\ln M) - N(\delta_{-1}(N) + \delta_0(N-1) + \delta_1(N-2)/2\cdot 1 + \cdots + \delta_{s-1}(N-s)/s(s-1)) + O(1)$; (b) $N^{-1}(N + K(N,1,M) - 2K(N,2,M) + \cdots + (-1)^{s-1}sK(N,s,M)) = (\ln N + \gamma - H_{s-1})/\ln M + 1/2 - (\delta_0(N-1) + \delta_1(N-2)/1 + \cdots + \delta_{s-1}(N-s)/(s-1)) + O(N^{-1})$; (c) $N^{-1}(N + (1-M^{-1})\times\sum_{k=2}^s(-1)^k\binom{k}{2}K(N,k,M)) = 1 + \frac12(1-M^{-1})((s-1)/\ln M + \delta_1(N-2) + \cdots + \delta_{s-1}(N-s)) + O(N^{-1})$.

21. Пусть всего имеется A_N узлов. Число непустых ссылок равно $A_N - 1$, а количество узлов без ссылок — N, так что общее количество пустых ссылок составляет $MA_N - A_N + 1 - N$. Для получения среднего количества пустых ссылок в любом фиксированном узле следует разделить найденное значение на M. [Среднее значение A_N приведено в упр. 20, (a).]

22. Для каждой из M^l последовательностей лидирующих битов имеется такой узел, что, по меньшей мере, два ключа начинаются с этой последовательности. Вероятность того, что с нее начинается ровно k ключей, составляет

$$\binom{N}{k}M^{-lk}(1-M^{-l})^{N-k},$$

так что среднее количество узлов луча на уровне l равно $M^l(1-(1-M^{-l})^N) - N(1-M^{-l})^{N-1}$.

23. Рассмотрим более общую задачу — случай для произвольного s (как в упр. 20). Если на уровне l имеется a_l узлов, то в них содержится a_{l+1} ссылок и $Ma_l - a_{l+1}$ позиций, в которых поиск может быть неудачным. Таким образом, среднее количество проверок цифр составляет $\sum_{l\ge 0}(l+1)M^{-l-1}(Ma_l - a_{l+1}) = \sum_{l\ge 0} M^{-l}a_l$. Используя формулу для a_l в случайном луче, получаем

$$1 + \frac{K(N{+}1,1,M) - 2K(N{+}1,2,M) + \cdots + (-1)^s(s{+}1)K(N{+}1,s{+}1,M)}{N+1}$$
$$= \frac{\ln N + \gamma - H_s}{\ln M} + \frac{1}{2} - \delta_0(N) - \frac{\delta_1(N{-}1)}{1} - \cdots - \frac{\delta_s(N{-}s)}{s} + O(N^{-1}).$$

24. Необходимо найти решения рекуррентных соотношений $x_0 = x_1 = y_0 = y_1 = 0$,

$$x_n = m^{-n} \sum_{n_1+\cdots+n_m=n} \binom{n}{n_1,\ldots,n_m} \Bigl(x_{n_1} + \cdots + x_{n_m} + \sum_{1\le j\le m} [n_j \ne 0]\Bigr)$$
$$= a_n + m^{1-n} \sum_k \binom{n}{k} x_k,$$

$$y_n = m^{-n} \sum_{n_1+\cdots+n_m=n} \binom{n}{n_1,\ldots,n_m} \Bigl(y_{n_1} + \cdots + y_{n_m} + \sum_{1\le i<j\le m} [n_i \ne 0] n_j\Bigr)$$
$$= b_n + m^{1-n} \sum_k \binom{n}{k} y_k,$$

для $n \ge 2$, где $a_n = m(1-(1-1/m)^n)$ и $b_n = \frac{1}{2}(m-1)n(1-(1-1/m)^{n-1})$. Согласно упр. 17 и 18 ответы таковы: (a) $x_N = N + V_N - U_N - [N=1] = A_N + N - 1$ (этот результат может быть получен непосредственно, поскольку количество узлов в лесу всегда на $N-1$ больше количества узлов в соответствующем луче!); (b) $y_N/N = \frac{1}{2}(M-1)V_N/N = \frac{1}{2}(M-1)((\ln N+\gamma)/\ln M - \frac{1}{2} - \delta_0(N-1)) + O(N^{-1})$.

25. (a) Пусть $A_N = M(N-1)/(M-1) - E_N$; тогда при $N \ge 2$ имеем $(1-M^{1-N})E_N = M - 1 - M(1-1/M)^{N-1} + M^{1-N}\sum_{0<k<N}\binom{N}{k}(M-1)^{N-k}E_k$. Поскольку $M-1 \ge M(1-1/M)^{N-1}$, по индукции находим, что $E_N \ge 0$. (b) По теореме 1.2.7A при $x = 1/(M-1)$ и $n = N-1$ находим $D_N = a_N + M^{1-N}\sum_k \binom{N}{k}(M-1)^{N-k}D_k$, где $a_1 = 0$ и $0 < a_N < M(1-1/M)^N/\ln M \le (M-1)^2/M\ln M$ при $N \ge 2$. Следовательно, $0 \le D_N \le (M-1)^2 A_N/M\ln M \le (M-1)(N-1)/\ln M$.

26. Приняв $q = \frac{1}{2}$, $z = -\frac{1}{2}$ во втором тождестве упр. 5.1.1–16, получим $1/3 - 1/(3\cdot 7) + 1/(3\cdot 7\cdot 15) - \cdots = 0.28879$; несколько удобнее использовать $z = -\frac{1}{4}$ и взять половину полученного результата. Можно также применить формулу Эйлера из упр. 5.1.1–14, включающую только отрицательные степени двойки. (Джон Ренч (John Wrench) вычислил это значение с точностью до 40 десятичных цифр: 0.28878 80950 86602 42127 88997 21929 23078 00889+.)

27. (Ради собственного удовольствия доведем точность решения до $O(N^{-1})$.) В обозначениях из упр. 5.2.2–38 и 5.2.2–48 имеем

$$\bar{C}_N = U_N + N - 1 + \frac{V_{N+1}}{N+1} - \alpha N - \beta + \sum_{n\ge 2}(-1)^n 2^{-n(n+1)/2} \frac{\sum_{m\ge 0}(2^{1-n})^m(1-2^{-m})^N}{\prod_{r=1}^n(1-2^{-r})},$$

где

$$\alpha = 2/(1\cdot 1) - 4/(3\cdot 3\cdot 1) + 8/(7\cdot 7\cdot 3\cdot 1) - 16/(15\cdot 15\cdot 7\cdot 3\cdot 1) + \cdots$$
$$\approx 1.60669\,51524\,15291\,76378\,33015\,23190\,92458\,04806-,$$

а $\beta = 2/(1\cdot 3\cdot 1) - 4/(3\cdot 7\cdot 3\cdot 1) + 8/(7\cdot 15\cdot 7\cdot 3\cdot 1) - \cdots \approx 0.60670$. Эти численные оценки приводят к выводу, что $\alpha = \beta + 1$, т. е. к факту, который нетрудно доказать. Значение

$\sum_{m\ge 0}(2^{1-n})^m(1-2^{-m})^N$ равно $O(N^{1-n})$ согласно упр. 5.2.2–46; а $V_{N+1}/(N+1)=U_{N+1}-U_N$. Следовательно, $\bar{C}_N=U_{N+1}-(\alpha-1)N-\alpha+O(N^{-1})=(N+1)\lg(N+1)+N\big((\gamma-1)/\ln 2+\frac{1}{2}-\alpha+\delta_{-1}(N)\big)+\frac{1}{2}-1/\ln 4-\alpha-\frac{1}{2}\delta_1(N)+O(N^{-1})$ согласно упр. 5.2.2–50.

Отклонение длины внутреннего пути дерева цифрового поиска было вычислено в работе Kirschenhofer, Prodinger, and Szpankowski, *SICOMP* **23** (1994), 598–616.

28. Выкладки в тексте и упр. 27 применимы для любого $M\ge 2$ — следует только подставить M вместо 2 в соответствующих, вполне очевидных местах. Следовательно, среднее количество проверок цифр при случайном успешном поиске составляет $\bar{C}_N/N=U_{N+1}-\alpha_M+1+O(N^{-1})=\log_M N+(\gamma-1)/\ln M+\frac{1}{2}-\alpha_M+\delta_{-1}(N)+(\log_M N)/N+O(N^{-1})$; при неудачном поиске оно равно $\bar{C}_{N+1}-\bar{C}_N=V_{N+2}/(N+2)-\alpha_M+1+O(N^{-1})=\log_M N+\gamma/\ln M+\frac{1}{2}-\alpha_M-\delta_0(N+1)+O(N^{-1})$. Здесь $\delta_s(n)$ определена в упр. 19, а

$$\alpha_M=\sum_{j\ge 0}(-1)^j M^{j+1}/(M^{j+1}-1)^2(M^j-1)\ldots(M-1).$$

29. Флажоле (Flajolet) и Седгевик (Sedgewick) [*SICOMP* **15** (1986), 748–767] показали, что среднее количество таких узлов примерно равно $0.372N$ при $M=2$ и $0.689N$ при $M=16$. В работе Флажоле и Ричмонда [*Random Structures and Algorithms* **3** (1992), 305–320] приводится обобщение этого результата.

30. Итерируя рекуррентное соотношение, получаем $h_n(z)$ в виде суммы всех возможных членов вида

$$\binom{n}{p_1}\frac{z}{2^{p_1}-1}\binom{p_1}{p_2}\frac{z}{2^{p_2}-1}\cdots\frac{z}{2^{p_m}-1}\binom{p_m}{1}\quad\text{для } n>p_1>\cdots>p_m>1.$$

31. $h_n'(1)=V_n$; см. упр. 5.2.2–36, (b). [Дополнительную информацию о дисперсии и предельных распределениях M-арных обобщений деревьев метода "Патриция" можно найти в работах P. Kirschenhofer and H. Prodinger, *Lecture Notes in Comp. Sci.* **226** (1986), 177–185; W. Szpankowski, *JACM* **37** (1990), 691–711; B. Rais, P. Jacquet, and W. Szpankowski, *SIAM J. Discrete Math.* **6** (1993), 197–213.]

32. Сумма полей SKIP равна количеству узлов в соответствующем бинарном дереве, поэтому ответом является величина A_N (см. упр. 20).

33. Вот как была получена формула (18). $A(2z)-2A(z)=e^{2z}-2e^z+1+A(z)(e^z-1)$ может быть приведено к виду $A(2z)/(e^{2z}-1)=(e^z-1)/(e^z+1)+A(z)/(e^z-1)$. Следовательно, $A(z)=(e^z-1)\sum_{j\ge 1}(e^{z/2^j}-1)/(e^{z/2^j}+1)$. Теперь, если $f(z)=\sum c_n z^n$, то $\sum_{j\ge 1}f(z/2^j)=\sum c_n z^n/(2^n-1)$. В нашем случае $f(z)=(e^z-1)/(e^z+1)=\tanh(z/2)$, что эквивалентно $1-2z^{-1}(z/(e^z-1)-2z/(e^{2z}-1))=\sum_{n\ge 1}B_{n+1}z^n(2^{n+1}-1)/(n+1)!$. Дальнейший вывод очевиден.

34. (a) Рассмотрим $\sum_{j\ge 1}\sum_{k=2}^{n-1}\binom{n}{k}B_k/2^{j(k-1)}$; согласно упр. 1.2.11.2–4 $1^{n-1}+\cdots+(m-1)^{n-1}=(B_n(m)-B_n)/n$. (b) Пусть $S_n(m)=\sum_{k=1}^{m-1}(1-k/m)^n$ и $T_n(m)=1/(e^{n/m}-1)$. При $k\le m/2$ имеем $e^{-kn/m}>\exp(n\ln(1-k/m))>\exp(-kn/m-k^2n/m^2)>e^{-kn/m}(1-k^2/m^2)$. Значит, $(1-k/m)^n=e^{-kn/m}+O(e^{-kn/m}k^2n/m^2)$. Поскольку $S_n(m)=\sum_{k=1}^{m/2}(1-k/m)^n+O(2^{-n})$ и $T_n(m)=\sum_{k=1}^{m/2}e^{-kn/m}+O(e^{-n/2})$, имеем $S_n(m)=T_n(m)+O(e^{-n/m}n/m^2)$. Сумма членов $O(\exp(-n/2^j)n/2^{2j})$ равна $O(n^{-1})$, так как сумма при $j\le\lg n$ имеет порядок $n^{-1}(1+2/e+(2/e)^2+\cdots$, а сумма при $j\ge\lg n$ — порядок $n^{-1}(1+1/4+(1/4)^2+\cdots$. (c) Доказательство аналогично доказательству, приведенному в разделе 5.2.2 при $|x|<2\pi$; затем используется аналитическое продолжение. (d) $\frac{1}{2}\lg(n/\pi)+\gamma/(2\ln 2)-\frac{3}{4}+\delta(n)+2/n$, где

$$\delta(n) = (2/\ln 2)\sum_{k\geq 1}\Re(\zeta(-2\pi ik/\ln 2)\Gamma(-2\pi ik/\ln 2)\exp(2\pi ik\lg n))$$
$$= (1/\ln 2)\sum_{k\geq 1}\Re(\zeta(1+2\pi ik/\ln 2)\exp(2\pi ik\lg(n/\pi)))/\cosh(\pi^2 k/\ln 2).$$

Дисперсия и высшие моменты были вычислены В. Шпанковским (W. Szpankowski), *JACM* **37** (1990), 691–711.

35. Ключи должны иметь вид $\{\alpha 0\beta 0\omega_1, \alpha 0\beta 1\omega_2, \alpha 1\gamma 0\omega_3, \alpha 1\gamma 1\delta 0\omega_4, \alpha 1\gamma 1\delta 1\omega_5\}$, где $\alpha, \beta, \ldots$ есть строки из нулей и единиц; $|\alpha| = a-1$, $|\beta| = b-1$ и т. д. Вероятность того, что пять случайных ключей имеют такой вид, равна $5!\,2^{a-1+b-1+c-1+d-1}/2^{a+b+a+b+a+c+a+c+d+a+c+d} = 5!/2^{4a+b+2c+d+4}$.

36. Пусть n — число внутренних узлов. (a) $(n!/2^I)\prod(1/s(x)) = n!\prod(1/2^{s(x)-1}s(x))$, где I — длина внутреннего пути дерева. (b) $((n+1)!/2^n)\prod(1/(2^{s(x)}-1))$. (Рассмотрите суммирование ответа к упр. 35 по всем a, b, c, $d \geq 1$.)

37. Наименьшая модифицированная длина внешнего пути равна $2 - 1/2^{N-2}$ и достигается только в случае вырожденного дерева (длина внешнего пути которого *максимальна*). (Можно доказать, что наибольшая модифицированная длина внешнего пути достигается тогда и только тогда, когда все внешние узлы расположены не более чем на двух смежных уровнях! Однако дерево с меньшей длиной внешнего пути не всегда имеет бóльшую модифицированную длину внешнего пути.)

38. Рассмотрите подзадачу поиска деревьев с k узлами с параметрами $(\alpha, \beta), (\alpha, \frac{1}{2}\beta), \ldots, (\alpha, 2^{k-n}\beta)$.

39. См. Miyakawa, Yuba, Sugito, and Hoshi, *SICOMP* **6** (1977), 201–234.

40. Пусть N/r — истинная длина периода последовательности. Построим дерево, подобное дереву метода "Патриция", с $a_0a_1\ldots$ в качестве массива TEXT и с N/r ключами, начинающимися с позиций $0, 1, \ldots, N/r-1$. (В соответствии с выбором r ни один ключ не является началом другого.) Включим в каждый узел поле SIZE, в котором содержится количество помеченных полей ссылок в поддереве, лежащем ниже этого узла. Для выполнения указанной операции используйте алгоритм P. Если поиск неудачен, ответ — 0; в случае успешного поиска и $j \leq n$ ответ — r. И наконец, если поиск успешен и $j > n$, ответ равен $r \cdot$ SIZE(P).

43. Ожидаемая высота асимптотически приближается к $(1 + 1/s)\log_M N$ с отклонением $O(1)$. [См. H. Mendelson, *IEEE Transactions* **SE-8** (1982), 611–619; P. Flajolet, *Acta Informatica* **20** (1983), 345–369; L. Devroye, *Acta Informatica* **21** (1984), 229–237; B. Pittel, *Advances in Applied Probability* **18** (1986), 139–155; W. Szpankowski, *Algorithmica* **6** (1991), 256–277.]

Средняя высота случайного дерева цифрового поиска с $M = 2$ асимптотически приближается к $\lg n + \sqrt{2\lg n}$ [Aldous and Shields, *Probability Theory and Related Fields* **79** (1988), 509–542]; то же самое справедливо и для случайного дерева метода "Патриция" [Pittel and Rubin, *Journal of Combinatorial Theory* **A55** (1990), 292–312].

44. См. *SODA* 8 (1997), 360–369; такая структура поиска тесно связана с алгоритмом быстрого многоключевого поиска, обсуждавшимся в ответе к упр. 5.2.2-30. Ж. Клемент (J. Clément), Ф. Флажоле (P. Flajolet) и Б. Вали (B. Vallée) показали, что при тернарном представлении поиск по лучу выполняется примерно в три раза быстрее, чем при использовании бинарного представления (2), с учетом доступа к узлам [см. *SODA* **9** (1998), 531–539].

45. Вероятность {THAT, THE, THIS} перед {BUILT, HOUSE, IS, JACK}, {HOUSE, IS, JACK} перед {BUILT}, {HOUSE, IS} перед {JACK}, {IS} перед {HOUSE}, {THIS} перед {THAT, THE} и {THE} перед {THAT} равна $\frac{3}{7}\cdot\frac{3}{4}\cdot\frac{2}{3}\cdot\frac{1}{2}\cdot\frac{1}{3}\cdot\frac{1}{2} = \frac{1}{56}$.

РАЗДЕЛ 6.4

1. $-37 \leq$ rI1 ≤ 46. Таким образом, в ячейках памяти, предшествующих TABLE и следующих за TABLE, не должны содержаться данные, которые соответствуют аргументу

(например, их первый байт мог бы быть нулевым). Очевидно, что было бы плохо хранить K в таком диапазоне! (Значит, можно сказать, что метод из упр. 6.3–4 использует меньшее пространство, поскольку границы той таблицы никогда не нарушаются.)

2. TOW (буксир). (В состоянии ли читатель найти десять слов из не более чем пяти букв, которые заполнят пустоты между −10 и 30?)

3. Коды символов удовлетворяют соотношениям A + T = I + N, B − E = O − R, а потому получим либо $f(\mathtt{AT}) = f(\mathtt{IN})$, либо $f(\mathtt{BE}) = f(\mathtt{OR})$. Обратите внимание на то, как команды 4 и 5 из табл. 1 разрешают эту дилемму, оставляя rI1 в достаточно узком диапазоне.

4. Рассмотрим случай для k пар. Наименьшее n, такое, что

$$m^{-n} n! \sum_k \binom{m}{n-k}\binom{n-k}{k} 2^{-k} < \frac{1}{2} \qquad \text{при } m = 365,$$

равно 88. Если вы пригласите 88 человек (включая себя), вероятность появления трио с одним днем рождения составит 0.511065; для 87 человек эта вероятность снижается до 0.499455. См. C. F. Pinzka, *AMM* **67** (1960), 830.

5. Эта хеш-функция плоха, поскольку она принимает не более 26 различных значений, причем некоторые из них будут встречаться явно чаще других. Даже при двойном хешировании (если предположить, например, что $h_2(K) = 1 +$ *второй байт* K и $M = 101$) низкая скорость поиска превысит выигрыш от высокой скорости вычисления хеш-функции. Кроме того, поскольку в программах на языке FORTRAN часто встречается куда больше, чем 100 различных переменных, значение $M = 100$ явно мало́.

6. Не на MIX-компьютере, поскольку почти всегда будет получаться арифметическое переполнение (слишком велико делимое). (Было бы неплохо иметь возможность вычислить $(wK) \bmod M$, в особенности для линейного исследования при $c = 1$, но, к сожалению, большинство компьютеров не предоставляют ее из-за переполнения частного.)

7. Если $R(x)$ кратен $P(x)$, то $R(\alpha^j) = 0$ в $\mathrm{GF}(2^k)$ для всех $j \in S$. Пусть $R(x) = x^{a_1} + \cdots + x^{a_s}$, где $a_1 > \cdots > a_s \geq 0$ и $s \leq t$. Выберем $t - s$ значений $a_{s+1}, \ldots, a_t$, таких, что $a_1, \ldots, a_t$ представляют собой различные неотрицательные числа, меньшие n. Матрица Вандермонда

$$\begin{pmatrix} \alpha^{a_1} & \ldots & \alpha^{a_t} \\ \alpha^{2a_1} & \ldots & \alpha^{2a_t} \\ \vdots & & \vdots \\ \alpha^{ta_1} & \ldots & \alpha^{ta_t} \end{pmatrix}$$

сингулярна, поскольку сумма ее первых s столбцов равна нулю. Однако это противоречит тому факту, что $\alpha^{a_1}, \ldots, \alpha^{a_t}$ — различные элементы $\mathrm{GF}(2^k)$ (см. упр. 1.2.3–37).

(Первоначально идея полиномиального хеширования была предложена группой исследователей из IBM (M. Hanan, S. Muroga, F. P. Palermo, N. Raver, and G. Schay, *IBM J. Research & Development* **7** (1963), 121–129; см. также *U. S. Patent 3311888* (1967)).)

8. По индукции. Сильная индуктивная гипотеза может быть дополнена тем фактом, что $\{(-1)^k(rq_k + q_{k-1})\theta\} = (-1)^k(r(q_k\theta - p_k) + (q_{k-1}\theta - p_{k-1}))$ при $0 \leq r \leq a_k$. "Рекордно низкие" значения $\{n\theta\}$ достигаются для $n = q_1$, $q_2 + q_1$, $2q_2 + q_1$, $\ldots$, $a_2q_2 + q_1 = 0q_4 + q_3$, $q_4 + q_3$, $\ldots$, $a_4q_4 + q_3 = 0q_6 + q_5$, $\ldots$, "рекордно высокие" — для $n = q_0$, $q_1 + q_0$, $\ldots$, $a_1q_1 + q_0 = 0q_3 + q_2$, $\ldots$. Это шаги, на которых формируется интервал с номером 0 новой длины. (Дальнейшая структура может быть выведена обобщением системы счисления Фибоначчи из упр. 1.2.8–34; см. L. H. Ramshaw, *J. Number Theory* **13** (1981), 138–175.)

9. Имеем $\phi^{-1} = //1, 1, 1, \ldots//$ и $\phi^{-2} = //2, 1, 1, \ldots//$. Пусть $\theta = //a_1, a_2, \ldots//$, $\theta_k = //a_{k+1}, a_{k+2}, \ldots//$ и пусть $Q_k = q_k + q_{k-1}\theta_{k-2}$ в обозначениях из упр. 8. Если $a_1 > 2$, плохим является самое первое разбиение. Три размера интервалов в упр. 8 равны $(1 - r\theta_{k-1})/Q_k$,

θ_{k-1}/Q_k и $(1-(r-1)\theta_{k-1})/Q_k$ соответственно, так что отношение первых двух длин составляет $(a_k - r) + \theta_k$. Эта величина меньше $\frac{1}{2}$ при $r = a_k$ и $a_{k+1} \geq 2$; следовательно, чтобы не было плохих разбиений $\{a_2, a_3, \ldots\}$ должны быть равны 1. (Другие связанные с данной темой теоремы можно найти в работе R. L. Graham and J. H. van Lint, *Canadian J. Math.* **20** (1968), 1020–1024, и в указанной в ней литературе.)

10. Вы найдете элегантное доказательство в работе F. M. Liang, *Discrete Math.* **28** (1979), 325–326.

11. Проблемы могут начаться при $K = 0$. Если потребовать, чтобы все ключи были ненулевыми, как в программе L, такое изменение может иметь смысл; при этом можно было бы помечать пустые позиции нулевым значением.

12. Можно хранить K в KEY[0], заменив при этом строки 14–19 следующими строками.

```
   STA  TABLE(KEY)        A - S1          CMPA TABLE,2(KEY)  C - 1 - S2
   CMPA TABLE,2(KEY)      A - S1          JNE  2B            C - 1 - S2
   JE   3F                A - S1       3H J2Z  5F              A - S1
2H ENT1 0,2               C - 1 - S2      ENT1 0,2               S2
   LD2  TABLE,1(LINK)     C - 1 - S2      JMP  SUCCESS           S2  ▌
```

"Сохраненное" время составляет $C - 1 - 5A + S + 4S1$ единиц, что в результате оборачивается *потерей*, поскольку C очень редко превышает 5. (Оказывается, не всегда следует оптимизировать внутренние циклы!)

13. Пусть записи в таблице относятся к двум различным типам, как и в алгоритме C, с дополнительным однобитовым полем TAG[i] в каждой записи. В приведенном решении используются циклические списки, следуя предложению Аллена Ньювелла (Allen Newell), с установленным TAG[i] $= 1$ в первом слове каждого списка.

A1. [Инициализация.] Установить $i \leftarrow j \leftarrow h(K) + 1$, $Q \leftarrow q(K)$.

A2. [Это список?] Если TABLE[i] пуст, установить TAG[i] $\leftarrow 1$ и перейти к шагу A8. В противном случае, если TAG[i] $= 0$, перейти к шагу A7.

A3. [Сравнение.] Если $Q =$ KEY[i], алгоритм успешно завершается.

A4. [Переход к следующему.] Если LINK[i] $\neq j$, установить $i \leftarrow$ LINK[i] и вернуться к шагу A3.

A5. [Поиск пустого узла.] Уменьшить R один или несколько раз до тех пор, пока не будет найдено значение, такое, что TABLE[R] пуст. Если $R = 0$, алгоритм завершается после переполнения; в противном случае установить LINK[i] $\leftarrow R$.

A6. [Подготовка к вставке.] Установить $i \leftarrow R$, TAG[R] $\leftarrow 0$ и перейти к шагу A8.

A7. [Перемещение записи.] Установить $i \leftarrow$ LINK[i] один или несколько раз, пока не будет выполнено условие LINK[i] $= j$. Затем выполнить шаг A5 и установить TABLE[R] $\leftarrow$ TABLE[i], $i \leftarrow j$, TAG[j] $\leftarrow 1$.

A8. [Вставка нового ключа.] Пометить TABLE[i] как занятый узел с KEY[i] $\leftarrow Q$, LINK[i] $\leftarrow j$. ▌

(Заметьте, что, если TABLE[i] занят, можно определить соответствующий полный ключ K по данному значению i. Имеем $q(K) =$ KEY[i], а затем, несколько раз присвоив $i \leftarrow$ LINK[i] до тех пор, пока не выполнится равенство TAG[i] $= 1$, получим $h(K) = i - 1$.)

14. Согласно указанным соглашениям запись "X $\Leftarrow$ AVAIL" из 2.2.3-(6) теперь означает следующее. "Установить X $\leftarrow$ AVAIL, присвоить X $\leftarrow$ LINK(X) нуль или несколько раз до X $= \Lambda$ (ошибка переполнения) или до TAG(X) $= 0$ и наконец установить AVAIL $\leftarrow$ LINK(X)."

Для вставки нового ключа K: установить Q $\Leftarrow$ AVAIL, TAG(Q) $\leftarrow 1$ и сохранить K в этом слове. (Можно также, если все ключи коротки, опустить этот шаг и заменить

в дальнейших шагах Q на K.) Затем установить R $\Leftarrow$ AVAIL, TAG(R) $\leftarrow$ 1, AUX(R) $\leftarrow$ Q, LINK(R) $\leftarrow \Lambda$. Присвоить P $\leftarrow h(K)$, а затем,

если TAG(P) $= 0$, установить TAG(P) $\leftarrow 2$, AUX(P) $\leftarrow$ R;

если TAG(P) $= 1$, установить S $\Leftarrow$ AVAIL, CONTENTS(S) $\leftarrow$ CONTENTS(P), TAG(P) $\leftarrow 2$, AUX(P) $\leftarrow$ R, LINK(P) $\leftarrow$ S;

если TAG(P) $= 2$, установить LINK(R) $\leftarrow$ AUX(P), AUX(P) $\leftarrow$ R.

Для получения ключа K: установить P $\leftarrow h(K)$ и,

если TAG(P) $\neq 2$, K отсутствует;

если TAG(P) $= 2$, установить P $\leftarrow$ AUX(P), а затем присвоить P $\leftarrow$ LINK(P) нуль или несколько раз до тех пор, пока не выполнится либо P $= \Lambda$, либо TAG(P) $= 1$, тогда либо AUX(P) $= K$ (если все ключи коротки), либо AUX(P) указывает на слово, содержащее K (возможно, косвенно через слово с TAG $= 2$).

В оригинальной схеме Элькока [*Comp. J.* **8** (1965), 242–243] в действительности использовались TAG $= 2$ и TAG $= 3$ для того, чтобы различить списки единичной длины (когда можно сохранить одно слово памяти) и более длинные списки. Это улучшение заслуживает внимания, поскольку мы предположительно работаем с такой большой хеш-таблицей, что почти все списки имеют единичные длины.

Другой путь размещения хеш-таблицы "наверху" большой связанной памяти с использованием срастающихся списков вместо раздельных цепочек был предложен Дж. С. Виттером (J. S. Vitter) [*Inf. Proc. Letters* **13** (1981), 77–79].

15. Зная о том, что всегда имеется свободный узел, можно сделать внутренний цикл более быстрым, поскольку нет необходимости в счетчике для определения, сколько раз был выполнен шаг L2. Более короткая программа компенсирует одну потерянную ячейку. (С другой стороны, в алгоритме L изящно указывается, как избежать использования переменной N и допустить полное заполнение таблицы без заметного замедления работы метода (за исключением случая реального переполнения таблицы): просто проверка $i < 0$ должна выполняться дважды! К алгоритму D этот трюк неприменим.)

16. Нет: 0 всегда приводит к метке SUCCESS, независимо от того, был ли он вставлен. В разные моменты мы попадаем на метку SUCCESS с различными значениями i.

17. Тогда вторая проба всегда будет обращаться к позиции 0.

18. "Стоимость" кода (31) на $3(A - S1)$ единиц больше, чем стоимость кода (30); экономия при этом составляет $4u$, умноженное на разность между (26), (27) и (28), (29). В случае успешного поиска (31) предпочтительнее только тогда, когда таблица заполнена более чем на 94%; выигрыш при этом не превышает $\frac{1}{2}u$. В случае неудачного поиска (31) предпочтительнее, если таблица заполнена более чем на 71%.

20. Мы хотим показать, что

$$\binom{j}{2} \equiv \binom{k}{2} \text{ (по модулю } 2^m) \qquad \text{и} \qquad 1 \le j \le k \le 2^m$$

влечет $j = k$. Заметим, что тождество $j(j-1) \equiv k(k-1)$ (по модулю 2^{m+1}) приводит к $(k-j)(k+j-1) \equiv 0$. Если $k - j$ нечетно, $k + j - 1$ должно быть кратно 2^{m+1}, но это невозможно, поскольку $2 \le k + j - 1 \le 2^{m+1} - 2$. Следовательно, $k - j$ четно, так что $k + j - 1$ нечетно и $k - j$ кратно 2^{m+1}, откуда $k = j$. (И обратно, если M не является степенью 2, такая последовательность проб не будет работать.)

Эта последовательность проб имеет вторичную кластеризацию и приводит к увеличению времени работы программы D (модифицированной в соответствии с (30)) примерно на $\frac{1}{2}(C-1) - (A - S1)$ единиц, поскольку $B \approx \binom{C+1}{3}/M$ можно не принимать во внимание. Пока таблица не заполнена примерно на 60%, это дает небольшое улучшение работы.

21. При уменьшении N алгоритм D может некорректно работать, так как, достигнув состояния, в котором отсутствует пустое пространство, он зациклится. С другой стороны, если N не будет уменьшаться, алгоритм D может сообщить о переполнении при имеющемся свободном пространстве. Причем этот вариант — меньшее из двух зол, поскольку для освобождения от удаленных ячеек можно воспользоваться рехешированием (в этом случае алгоритм D должен увеличивать N и проверять переполнение только при вставке элемента в *пустую* позицию, так как N — количество непустых позиций). В программе также можно поддерживать два счетчика.

22. Предположим, что позиции $j-1, j-2, \ldots, j-k$ заняты, а $j-k-1$ пуста по модулю M. Ключи, которые проверяют позицию j и находят ее занятой перед вставкой, совпадают с ключами в позициях от $j-1$ до $j-k$, хеш-адреса которых не лежат между $j-1$ и $j-k$; такие "проблематичные" ключи появляются в порядке вставки. Алгоритм R перемещает первый такой ключ в позицию j и повторяет процесс на меньшем диапазоне проблематичных позиций до тех пор, пока будет оставаться хоть один проблематичный ключ.

23. Схема удаления для срастающихся цепочек, изобретенная Дж. С. Виттером (J. S. Vitter) [*J. Algorithms* **3** (1982), 261–275], сохраняет распределение времен поиска.

24. $P(P-1)(P-2)P(P-1)P(P-1)/(MP(MP-1)\ldots(MP-6)) = M^{-7}(1-(5-21/M)P^{-1}+O(P^{-2}))$. В общем, вероятность появления хеш-последовательности $a_1\ldots a_N$ равна $(\prod_{j=0}^{M-1}\times \times P^{b_j})/(MP)^{\underline{N}} = M^{-N}+O(P^{-1})$, где b_j — количество a_i, равных j.

25. Пусть $(N+1)$-й ключ хешируется в позицию a; P_k равно M^{-N}, умноженному на количество хеш-последовательностей, которые оставляют k позиций $a, a-1, \ldots, a-k+1$ (по модулю M) занятыми, а $a-k$ — пустой. Количество таких последовательностей с занятыми $a+1, \ldots, a+t$ и пустой $a+t+1$ равно $g(M,N,t+k)$ в силу циклической симметрии алгоритма.

26. $\dfrac{9!}{2!\,2!\,4!\,1!}f(3,2)f(3,2)f(5,4)f(2,1) = 2^2 3^5 5^4 7 = 4\,252\,500.$

27. Следуя указанию, находим

$$s(n,x,y) = \sum_k \binom{n}{k} x(x+k)^k (y-k)^{n-k-1}(y-n) + n\sum_k \binom{n-1}{k-1}(x+k)^k(y-k)^{n-k-1}(y-n).$$

В первой сумме заменим k на $n-k$ и применим формулу Абеля; во второй заменим k на $k+1$. Теперь

$$g(M,N,k) = \binom{N}{k}(k+1)^{k-1}(M-k-1)^{N-k-1}(M-N-1)$$

с $0/0=1$ при $k=N=M-1$ и

$$\begin{aligned} M^N\sum(k+1)P_k &= \sum_{k\ge 0}\binom{k+2}{2}g(M,N,k) \\ &= \frac{1}{2}\left(\sum_{k\ge 0}(k+1)g(M,N,k) + \sum_{k\ge 0}(k+1)^2 g(M,N,k)\right). \end{aligned}$$

Первая сумма равна $M^N\sum P_k = M^N$, вторая — $s(N,1,M-1) = M^N + 2NM^{N-1} + 3N(N-1)M^{N-2} + \cdots = M^N Q_1(M,N)$. (См. J. Riordan, *Combinatorial Identities* (New York: Wiley, 1968), 18–23; здесь приводится дополнительная информация о суммах наподобие $s(n,x,y)$.)

28. Пусть $t(n,x,y) = \sum_{k\ge 0}\binom{n}{k}(x+k)^{k+2}(y-k)^{n-k-1}(y-n)$; тогда, как и в упр. 27, находим $t(n,x,y) = xs(n,x,y) + nt(n-1,x+1,y-1)$, $t(N,1,M-1) = M^N(3Q_3(M,N) - 2Q_2(M,N))$. Значит, $\sum(k+1)^2P_k = M^{-N}\sum(\frac{1}{3}(k+1)^3 + \frac{1}{2}(k+1)^2 + \frac{1}{6}(k+1))g(M,N,k) = Q_3(M,N) -$

$\frac{2}{3}Q_2(M,N) + \frac{1}{2}Q_1(M,N) + \frac{1}{6}$. Вычитание $(C'_N)^2$ дает дисперсию, приближенно равную $\frac{3}{4}(1-\alpha)^{-4} - \frac{2}{3}(1-\alpha)^{-3} - \frac{1}{12}$. Стандартное отклонение зачастую больше среднего значения, например при $\alpha = .9$ среднее значение равно 50.5, а стандартное отклонение — $\frac{1}{2}\sqrt{27333} \approx 82.7$.

29. Пусть $M = m+1$, $N = n$; последовательность парковки та же, что и при применении алгоритма L к хеш-последовательности $(M - a_1)\ldots(M - a_n)$, при которой позиция 0 остается пустой. Следовательно, искомый ответ — $f(m+1, n) = (m+1)^n - n(m+1)^{n-1}$. [Эта задача была поставлена в работе A. G. Konheim and B. Weiss, *SIAM J. Applied Math.* **14** (1966), 1266–1274; см. также R. Pyke, *Annals of Math. Stat.* **30** (1959), 568–576, лемма 1.]

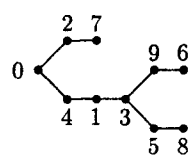
30. Очевидно, что если машины припарковались, то они определяют такую перестановку. И обратно, если имеется перестановка $p_1p_2\ldots p_n$, пусть $q_1q_2\ldots q_n$ означает обратную перестановку ($q_i = j$ тогда и только тогда, когда $p_j = i$) и пусть b_i — количество a_j, равных i. Каждый автомобиль припаркуется, если мы докажем, что $b_n \le 1$, $b_{n-1} + b_n \le 2$ и т. д. (что эквивалентно $b_1 \ge 1$, $b_1 + b_2 \ge 2$ и т. д.). Но это, очевидно, истинно, поскольку все k элементов $a_{q_1}, \ldots, a_{q_k}$ не больше k.

[Пусть r_j означает "левое влияние" q_j, а именно $r_j = k$ тогда и только тогда, когда $q_{j-1} < q_j, \ldots, q_{j-k-1} < q_j$ и либо $j = k$, либо $q_{j-k} > q_j$. Из всех перестановок $p_1\ldots p_n$, мажорирующих данную последовательность пробуждений $a_1\ldots a_n$, алгоритм "немедленно остановись!" находит наименьшую (в лексикографическом порядке). Конхейм и Вейсс заметили, что количество последовательностей пробуждений, приводящих к данной перестановке $p_1\ldots p_n$, равно $\prod_{j=1}^n r_j$; интересно, что сумма этих произведений, взятая по всем перестановкам $q_1\ldots q_n$, равна $(n+1)^{n-1}$.]

31. Таких возможных связей много, но следующие три — любимые автором [см. также Foata and Riordan, *Æquat. Math.* **10** (1974), 10–22].

a) В обозначениях из предыдущего ответа счетчики $b_1, b_2, \ldots, b_n$ соответствуют полной последовательности парковок тогда и только тогда, когда $(b_1, b_2, \ldots, b_n, 0)$ представляет собой корректную последовательность *степеней* узлов дерева в прямом порядке. (Сравните с 2.3.3–(9), иллюстрирующим обратный порядок.) Каждое такое дерево соответствует $n!/b_1!\ldots b_n!$ различным помеченным свободным деревьям на $\{0, \ldots, n\}$, поскольку можно пометить корень нулем, а для $k = 1, 2, \ldots, n$ последовательно в прямом порядке выбирать метки из оставшихся неиспользованными, из дочерних узлов k-го узла $(b_k + \cdots + b_n)!/b_k!\,(b_{k+1} + \cdots + b_n)!$ способами, назначая метки слева направо в порядке возрастания. Каждая такая последовательность счетчиков соответствует $n!/b_1!\ldots b_n!$ последовательностям пробуждения.

b) Доминик Фоата (Dominique Foata) дал следующее красивое взаимно однозначное соответствие. Пусть $a_1\ldots a_n$ — удачная последовательность парковок, при которой машина q_j располагается в позиции j. Помеченное свободное дерево на $\{0, 1, \ldots, n\}$ строится с помощью линий, проведенных из j в 0 при $a_j = 1$ и из j в $q_{a_j - 1}$ в противном случае для $1 \le j \le n$. (Будем считать узлы дерева автомобилями; автомобиль j связан с автомобилем, который припарковался именно в том месте, где проснулась j-я жена.) Например, моменты пробуждений 3 1 4 1 5 9 2 6 5 приводят согласно правилу Фоата к построению свободного дерева

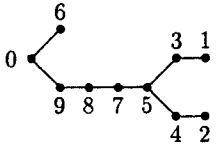


И обратно: последовательность припаркованных автомобилей может быть получена из дерева методом топологической сортировки в предположении, что стрелки указывают от

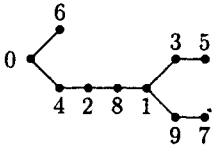
корневого нуля и выбирается наименьший "источник" на каждом шаге. Из этой последовательности можно восстановить $a_1 \ldots a_n$.

c) Сначала построим вспомогательное дерево, в котором родительский узел узла k является первым элементом, большим k, следующим за ним в перестановке $q_1 \ldots q_n$; если такого элемента нет, родительский узел становится узлом 0. Затем создадим копию вспомогательного дерева и повторно пометим ненулевые узлы нового дерева в прямом порядке при помощи следующей процедуры: если метка текущего узла во вспомогательном дереве была k, замените его текущую метку меткой, которая в текущий момент является $(1+p_k-a_k)$-й наименьшей в его поддереве. Например:

Вспомогательное дерево



Окончательное дерево



Для обращения процедуры можно перестроить вспомогательное дерево, выполняя в прямом порядке обмен меток каждого узла с наибольшей меткой в его поддереве.

Конструкции (a) и (b) сильно связаны, но конструкция (c) несколько отличается от них. Она имеет интересное свойство, заключающееся в том, что сумма перестановок автомобилей из их предпочтительных положений равна числу *инверсий* в дереве — числу пар меток $a > b$, где a является предком b. Эта связь между последовательностью парковок и инверсиями дерева была впервые открыта в работе G. Kreweras, *Periodica Math. Hung.* **11** (1980), 309–320. Тот факт, что инверсии деревьев имеют прямое отношение к связным графам [Mallows and Riordan, *Bull. Amer. Math. Soc.* **74** (1968), 92–94], позволяет сделать вывод о том, что сумма $\binom{D(p)}{k}$, взятая по всем последовательностям парковки, где $D(p) = (p_1 - a_1) + \cdots + (p_n - a_n)$, равна общему количеству связных графов с $n+k$ ребрами на множестве помеченных вершин $\{0, 1, \ldots, n\}$. [См. формулы (2.11), (3.5) и (8.13) в статье Janson, Łuczak, Knuth, and Pittel, *Random Struct. & Alg.* **4** (1993), 233–358.]

32. Будем считать индексы циклическими, т. е. $c_M = c_0$, $c_{M+1} = c_1$ и т. д. При $c_j = b_j + c_{j+1} - 1$ для всех j решений не существует, поскольку сумма по всем j дает $\sum c_j = \sum b_j + \sum c_j - M < \sum c_j$. Следовательно, каждое решение имеет $M - \sum b_j$ значений j, таких, что $b_j = c_{j+1} = 0$. Если $(c'_0, \ldots, c'_{M-1})$ представляет собой другое решение, то должно выполниться условие $c'_{j+1} > 0$ для хотя бы одного такого j; но отсюда вытекает противоречие: $c'_{j+2} > c_{j+2}$, $c'_{j+3} > c_{j+3}, \ldots$. Решение может быть найдено путем определения c_{M-1}, c_{M-2}, $\ldots$, если предположить, что $c_0 = 0$; тогда, если $c_0 > 0$, достаточно переопределять c_{M-1}, $c_{M-2}, \ldots$, пока дальнейшие изменения не станут ненужными.

33. Отдельные вероятности не являются независимыми, поскольку не было принято во внимание условие $b_0 + b_1 + \cdots + b_{M-1} = N$; это отклонение позволяет сумме $\sum b_j$ иметь любое данное неотрицательное значение с ненулевой вероятностью. Соотношения (46) не являются абсолютно корректными; из них следует, например, что q_k положительно при всех k, а это противоречит тому факту, что c_j никогда не превосходит $N-1$.

Гастон Гонне (Gaston Gonnet) и Дж. Мунро (J. Munro) [*J. Algorithms* **5** (1984), 451–470] обнаружили интересный путь вывода точного результата из аргумента, который привел к (51), введя полезную операцию, названную *преобразованием Пуассона* последовательности $\langle A_{mn} \rangle$: мы имеем $e^{-mz} \sum_n A_{mn} (mz)^n / n! = \sum_k a_k z^k$ тогда и только тогда, когда $A_{mn} = \sum_k a_k n^{\underline{k}} / m^k$.

34. (a) Существует $\binom{N}{k}$ способов выбора множества j, таких, что a_j имеет определенное значение, и $(M-1)^{N-k}$ способов присвоения значения другим a. Таким образом,

$$P_{Nk} = \binom{N}{k}(M-1)^{N-k}/M^N.$$

(b) $P_N(z) = B(z)$ в (50). (c) Рассмотрим общее количество проб для поиска всех ключей, не считая получения указателя на заголовок списка на рис. 38 (при использовании такой таблицы). Список длиной k позволяет добавить к общей сумме $\binom{k+1}{2}$ проб; следовательно,

$$C_N = M\sum\binom{k+1}{2}P_{Nk}/N = (M/N)(\tfrac{1}{2}P_N''(1) + P_N'(1)).$$

(d) В случае (i) для списка длиной k требуется k проб (не считая получения заголовка списка), в то время как в случае (ii) требуется $k+\delta_{k0}$ проб. Значит, в случае (ii) получим $C_N' = \sum(k+\delta_{k0})P_{Nk} = P_N'(1) + P_N(0) = N/M + (1-1/M)^N \approx \alpha + e^{-\alpha}$, а в случае (i) — просто $C_N' = N/M = \alpha$. В случае (iii) применима формула $MC_N' = M - N + NC_N$, поскольку $M-N$ хеш-адресов будут указывать на пустые позиции в таблице, в то время как N хеш-адресов приведут к поиску до конца некоторого списка из точки внутри него; это дает (18).

35. (i) $\sum(1+\frac{1}{2}k-(k+1)^{-1})P_{Nk} = 1+N/(2M)-M(1-(1-1/M)^{N+1})/(N+1) \approx 1+\frac{1}{2}\alpha-(1-e^{-\alpha})/\alpha$. (ii) Добавьте $\sum\delta_{k0}P_{Nk} = (1-1/M)^N \approx e^{-\alpha}$ к результату (i). (iii) Если неудачный поиск начинается с j-го элемента списка длиной k, данный ключ имеет случайный порядок по отношению к другим k элементам, так что ожидаемая длина поиска составляет $(j\cdot 1+2+\cdots+(k+1-j)+(k+1-j))/(k+1)$. Суммирование по j дает $MC_N' = M-N+M\sum(k^3+9k^2+2k)P_{Nk}/(6k+6) = M-N+M(\frac{1}{6}N(N-1)/M^2+\frac{3}{2}N/M-1+(M/(N+1))(1-(1-1/M)^{N+1}))$; следовательно, $C_N' \approx \frac{1}{2}\alpha + \frac{1}{6}\alpha^2 + (1-e^{-\alpha})/\alpha$.

36. (i) $N/M - N/M^2$. (ii) $\sum(\delta_{k0}+k)^2P_{Nk} = \sum(\delta_{k0}+k^2)P_{Nk} = P_N(0) + P_N''(1) + P_N'(1)$. Вычитание $(C_N')^2$ дает ответ $(M-1)N/M^2 + (1-1/M)^N(1-2N/M-(1-1/M)^N) \approx \alpha + e^{-\alpha}(1-2\alpha-e^{-\alpha}) \le 1-e^{-1}-e^{-2} = 0.4968$. [Для структур данных (iii) требуется более сложный анализ, подобный приведенному в упр. 37.]

37. Пусть S_N — среднее значение $(C-1)^2$ и пусть все $M^N N$ выборов хеш-последовательности и ключей равновероятны. Тогда

$$\begin{aligned}
M^N N S_N &= \frac{1}{3}\sum\binom{N}{k_1,\ldots,k_M}\left(k_1(k_1-\tfrac{1}{2})(k_1-1)+\cdots+k_M(k_M-\tfrac{1}{2})(k_M-1)\right)\\
&= \frac{1}{3}M\sum_k\binom{N}{k}(M-1)^{N-k}k(k-\tfrac{1}{2})(k-1)\\
&= \frac{1}{3}MN(N-1)(N-2)\sum_k\binom{N-3}{k-3}(M-1)^{N-k}\\
&\qquad + \frac{1}{2}MN(N-1)\sum_k\binom{N-2}{k-2}(M-1)^{N-k}\\
&= \frac{1}{3}MN(N-1)(N-2)M^{N-3} + \frac{1}{2}MN(N-1)M^{N-2}.
\end{aligned}$$

Дисперсия при этом равна $S_N - ((N-1)/2M)^2 = (N-1)(N+6M-5)/12M^2 \approx \frac{1}{2}\alpha + \frac{1}{12}\alpha^2$.

В *CMath* §8.5 рассмотрена интересная связь между общей дисперсией, вычисленной здесь, и двумя другими видами дисперсии: дисперсией (по случайным хеш-таблицам) среднего числа проб (по всем наличным элементам) и средней (по случайным хеш-таблицам) дисперсией числа проб (по всем наличным элементам). Общая дисперсия всегда является

суммой двух других; и в этом случае дисперсия среднего числа проб составляет $(M-1)(N-1)/(2M^2N)$.

38. Среднее число проб составляет $\sum P_{Nk}(2H_{k+1}-2+\delta_{k0})$ при неудачном поиске и $(M/N)\times \times \sum P_{Nk}k(2(1+1/k)H_k-3)$ — при успешном согласно формулам 6.2.2–(5) и 6.2.2–(6). Эти суммы равны $2f(N)+2M(1-(1-1/M)^{N+1})/(N+1)+(1-1/M)^N-2$ и $2(M/N)f(N)+2f(N-1)+2M(1-(1-1/M)^N)/N-3$ соответственно, где $f(N)=\sum P_{Nk}H_k$. В упр. 5.2.1–40 говорится о том, что $f(N)=\ln\alpha+\gamma+E_1(\alpha)+O(M^{-1})$ при $N=\alpha M$, $M\to\infty$.

[Хеширование с деревьями было впервые предложено в работе P. F. Windley, *Comp. J.* **3** (1960), 84–88. Анализ в предыдущем параграфе показывает, что такой метод не настолько лучше обычного метода цепочек, чтобы оправдать введение лишних полей ссылок — списки для этого слишком коротки. Более того, когда M мало́, хеширование с деревьями не настолько лучше обычного поиска по дереву, чтобы оправдать затраты времени на хеширование.]

39. (Этот подход к анализу алгоритма C был предложен Дж. С. Виттером (J. S. Vitter).) Имеем $c_{N+1}(k)=(M-k)c_N(k)+(k-1)c_N(k-1)$ при $k\ge 2$ и, кроме того, $\sum kc_N(k)=NM^N$. Следовательно,

$$\begin{aligned}S_{N+1}&=\sum_{k\ge 2}\binom{k}{2}c_{N+1}(k)=\sum_{k\ge 2}\binom{k}{2}((M-k)c_N(k)+(k-1)c_N(k-1))\\&=\sum_{k\ge 1}\Big((M+2)\binom{k}{2}+k\Big)c_N(k)=(M+2)S_N+NM^N.\end{aligned}$$

В результате получаем $S_N=(N-1)M^{N-1}+(N-2)M^{N-2}(M+2)+\cdots+M(M+2)^{N-2}=\frac{1}{4}(M(M+2)^N-M^{N+1}-2NM^N)$.

Рассмотрим общее число проб в случае неудачного поиска, просуммированное по всем M значениям функции $h(K)$; каждый список длиной k вносит вклад $k+\delta_{k0}+\binom{k}{2}$ в общую сумму, следовательно, $M^{N+1}C'_N=M^{N+1}+S_N$.

40. Определим U_N так же, как и S_N в упр. 39, но с $\binom{k}{2}$, замененным на $\binom{k+1}{3}$. Найдем, что $U_{N+1}=(M+3)U_N+S_N+NM^N$, значит,

$$U_N=\tfrac{1}{36}(M^N(M-6N)-9M(M+2)^N+8M(M+3)^N).$$

Дисперсия равна $2U_N/M^{N+1}+C'_N-(C'_N)^2$, что примерно составляет

$$\tfrac{35}{144}-\tfrac{1}{12}\alpha-\tfrac{1}{4}\alpha^2+(\tfrac{1}{4}\alpha-\tfrac{5}{8})e^{2\alpha}+\tfrac{4}{9}e^{3\alpha}-\tfrac{1}{16}e^{4\alpha}$$

при $N=\alpha M$, $M\to\infty$. При $\alpha=1$ эта величина равна приблизительно 4.50, так что стандартное отклонение ограничено величиной 2.12.

41. Пусть V_N — средняя длина блока занятых ячеек на "верхнем" конце таблицы. Вероятность того, что длина блока равна k, составляет $A_{Nk}(M-1-k)^{\underline{N-k}}/M^N$, где A_{Nk} — число хеш-последовательностей (35), таких, что алгоритм C оставляет первые $N-k$ и последние k ячеек занятыми, и таких, что подпоследовательность $1\ 2\ldots N-k$ оказывается расположенной в порядке возрастания. Вследствие этого

$$\begin{aligned}M^NV_N&=\textstyle\sum_k kA_{Nk}(M-1-k)^{\underline{N-k}}=M^{N+1}-\sum_k(M-k)A_{Nk}(M-1-k)^{\underline{N-k}}\\&=M^{N+1}-(M-N)\textstyle\sum_k A_{Nk}(M-k)^{\underline{N-k}}=M^{N+1}-(M-N)(M+1)^N.\end{aligned}$$

Теперь $T_N=(N/M)(1+V_N-T_0-\cdots-T_{N-1})$, поскольку $T_0+\cdots+T_{N-1}$ представляет собой среднее число предыдущих операций уменьшения R, а N/M — вероятность его уменьшения на текущем шаге. Решение этого рекуррентного соотношения — $T_N=(N/M)(1+1/M)^N$. (Такая простая формула заслуживает более простого доказательства!)

42. $S1_N$ равно количеству элементов, вставленных при $A = 0$, деленному на N.

43. Пусть $N = \alpha M'$ и $M = \beta M'$ и пусть $e^{-\lambda} + \lambda = 1/\beta$, $\rho = \alpha/\beta$. Тогда $C_N \approx 1 + \frac{1}{2}\rho$ и $C'_N \approx \rho + e^{-\rho}$ при $\rho \le \lambda$; $C_N \approx \frac{1}{8\rho}(e^{2\rho-2\lambda} - 1 - 2\rho + 2\lambda)(3 - 2/\beta + 2\lambda) + \frac{1}{4}(\rho + 2\lambda - \lambda^2/\rho)$ и $C'_N \approx 1/\beta + \frac{1}{4}(e^{2\rho-2\lambda} - 1)(3 - 2/\beta + 2\lambda) - \frac{1}{2}(\rho - \lambda)$ при $\rho \ge \lambda$. Если $\alpha = 1$, получим минимальное $C_N \approx 1.69$ при $\beta \approx .853$; наименьшее $C'_N \approx 1.79$ получается при $\beta \approx .782$. Установив $\beta = .86$, можно получить близкую к оптимальной производительность для широкого диапазона α. Таким образом, помещение первых коллизий в область, не конфликтующую с хеш-адресами, оправдывается, даже если при меньшем диапазоне хеш-адресов получается больше коллизий. Приведенные результаты были получены Дж. С. Виттером (Jeffrey S. Vitter), *JACM* **30** (1983), 231–258.

44. (Рассмотренный здесь способ решения "в лоб" был найден автором в 1972 году; гораздо более элегантное решение М. С. Патерсона (M. S. Paterson) можно найти в книге Грина (Greene) и Кнута (Knuth) *Mathematics for the Analysis of Algorithms* (Birkhäuser Boston, 1980), §3.4. Патерсон нашел также важные методы упрощения других способов выполнения анализа, описанных в этом разделе.)

Пронумеруем позиции массива от 1 до m слева направо. Рассматривая множество всех $\binom{n}{k}$ последовательностей операций с k "p-шагами" и $n-k$ "q-шагами" как равновероятные, обозначим через $g(m, n+1, k, r)$ умноженную на $\binom{n}{k}$ вероятность того, что первые $r-1$ позиций становятся занятыми, а r-я осталась пуста. Таким образом, $g(m, l, k, r)$ равна умноженной на $(m-1)^{-(l-1-k)}$ сумме по всем конфигурациям

$$1 \le a_1 < \cdots < a_k < l, \qquad (c_1, \ldots, c_{l-1-k}), \qquad 2 \le c_i \le m,$$

вероятностей того, что первая пустая позиция — r, когда a_j-я операция представляет собой p-шаг, а оставшиеся $l-1-k$ операций представляют собой q-шаги, которые начинаются с выбора позиций $c_1, \ldots, c_{l-1-k}$ соответственно. Выполнив суммирование по всем конфигурациям при дополнительном условии, что a_j-я операция занимает позицию b_j при данных $1 \le b_1 < \cdots < b_k < r$, получим рекуррентное соотношение

$$g(m, l, k{+}1, r) = \sum_{\substack{a<l \\ b<r \\ 1\le b\le a}} \frac{(l-b-1)!}{(l-r)!}\frac{(m-r)!}{(m-b)!}(m-l+1)g(m, a, k, b);$$

$$g(m, l, 0, r) = \frac{(l-1)!}{(l-r)!}\frac{(m-r)!}{m!}(m-l+1)\left(P_l + [r \ne 1]\frac{m}{l-1}(1-P_l)\right),$$

где $P_l = (m/(m-1))^{l-1}$. Обозначив $G(m, l, k) = \sum_{r=1}^{l}(m+1-r)g(m, l, k, r)$, получим, что

$$G(m, l, k{+}1) = \frac{m-l+1}{m-l+2}\sum_{a=1}^{l-1} G(m, a, k); \qquad G(m, l, 0) = \frac{m-l+1}{m-l+2}(m + P_l).$$

Решением поставленной задачи является $m - \sum_{k=0}^{n} p^k q^{n-k} G(m, n{+}1, k)$, что после некоторых преобразований становится равным $m - ((m-n)/(m-n+1))(Q_n + mR + pSR)$, где

$$Q_j = P_{j+1}q^j,$$

$$R = \left(1 - \frac{p}{m+1}\right)\left(1 - \frac{p}{m}\right)\cdots\left(1 - \frac{p}{m-n+2}\right) = \prod_{j=0}^{n-1}\left(1 - \frac{p}{m+1-j}\right),$$

$$S = \frac{\left(1-\frac{1}{m+1}\right)Q_0}{\left(1-\frac{p}{m+1}\right)} + \frac{\left(1-\frac{1}{m}\right)Q_1}{\left(1-\frac{p}{m+1}\right)\left(1-\frac{p}{m}\right)} + \cdots + \frac{1-\left(\frac{1}{m-n+2}\right)Q_{n-1}}{R}$$

$$= \sum_{k=0}^{n-1} \frac{(1-1/(m+1-k))Q_k}{\prod_{j=0}^{k}(1-p/(m+1-j))}.$$

При $p = 1/m$, $Q_j = 1$ для всех j. Считая $w = m+1$, $n = \alpha w$, $w \to \infty$, найдем $\ln R = -(H_w - H_{w(1-\alpha)})p + O(p^2)$; следовательно, $R = 1 + w^{-1}\ln(1-\alpha) + O(w^{-2})$; аналогично $S = \alpha w + O(1)$. Таким образом, ответ будет таким: $(1-\alpha)^{-1} - 1 - \alpha - \ln(1-\alpha) + O(w^{-1})$.

Примечание. Более простая задача "С вероятностью p займем крайнюю слева позицию; в противном случае займем случайным образом выбранную пустую позицию" решается, если принять $P_j = 1$ в приведенных выше формулах, и ответом будет $m-(m+1)(m-n)R/(m-n+1)$. Чтобы получить C'_N для случайных проб с вторичной кластеризацией, установим $n = N$, $m = M$ и добавим 1 к приведенному выше ответу.

45. Да. См. L. Guibas, *JACM* **25** (1978), 544–555.

46. Определим числа $\left[\left[{n \atop k}\right]\right]$ для $k \geq 0$ согласно правилу

$$\sum_k \binom{x+k}{k}\left[\left[{n \atop k}\right]\right] = (x+n+1)^n$$

для всех x и всех целых неотрицательных n. Полагая $x = -1, -2, \ldots, -n-1$, получим, что

$$\left[\left[{n \atop k}\right]\right] = \sum_j \binom{k}{j}(-1)^j(n-j)^n \qquad \text{для } 0 \leq k \leq n.$$

Затем, рассматривая $x = 0$, получим, что при $k > n$ можно считать $\left[\left[{n \atop k}\right]\right] = 0$, так что обе части определяющего уравнения представляют собой полиномы от x степени n, совпадающие в $n+1$ точке. Отсюда следует, что числа $\left[\left[{n \atop k}\right]\right]$ обладают указанным свойством.

Пусть $f(N,r)$ — количество хеш-последовательностей $a_1 \ldots a_N$, при которых первые r позиций заняты, а следующая — пустая. Существует $\binom{M-r-1}{N-r}$ способов размещения занятых ячеек, и каждый из них встречается столько раз, сколько имеется последовательностей $a'_1 \ldots a'_N$, $1 \leq a'_i \leq N$, таких, что в каждой последовательности каждое из чисел $r+1$, $r+2$, $\ldots$, N содержится хотя бы один раз. В соответствии с принципом включения-исключения имеется $\left[\left[{N \atop N-r}\right]\right]$ таких последовательностей. Таким образом,

$$f(N,r) = \binom{M-r-1}{N-r}\left[\left[{N \atop N-r}\right]\right].$$

Теперь

$$C'_N = 1 + M^{-N-1}\sum_{r=0}^{N} f(N,r)\left(\sum_{a=0}^{r-1} r + \sum_{a=r+1}^{M-1}\frac{N-r}{M-r-1}(r+1)\right)$$

$$= 1 + M^{-N-1}\sum_{r=0}^{N} f(N,r)(N+(N-1)r).$$

Полагая $S_n(x) = \sum_k k\binom{x+k}{k}\left[\left[{n \atop k}\right]\right]$, имеем

$$(x+1)^{-1}S_n(x) + \sum_k \binom{x+k}{k}\left[\left[{n \atop k}\right]\right] = \sum_k \binom{x+1+k}{k}\left[\left[{n \atop k}\right]\right];$$

следовательно, $S_n(x) = (x+1)((x+n+2)^n - (x+n+1)^n)$. Отсюда вытекает, что $C'_N = N(1+1/M) - (N-1)(1-N/M)(1+1/M)^N \approx N(1-(1-\alpha)e^{\alpha})$ и $C_N = (N-1)((1+$

$1/M)/2 + (1+1/M)^N) + (3M^2+6M+2)((1+1/M)^N - 1)/N - (3M+2)(1+1/M)^N$, что равно $(e-2.5)M + O(1)$ при $N = M-1$.

О других свойствах чисел $[\![{n \atop k}]\!]$ можно прочесть в книге John Riordan, *Combinatorial Identities* (New York: Wiley, 1968), 228–229.

47. Практически так же применим анализ алгоритма L! Любая последовательность проб с циклической симметрией, которая проверяет только соседние с ранее проверенными позиции, будет вести себя точно так же.

48. $C'_N = 1 + p + p^2 + \cdots$, где $p = N/M$ представляет собой вероятность того, что случайная позиция заполнена; следовательно, $C'_N = M/(M-N)$ и $C_N = N^{-1}\sum_{k=0}^{N-1} C'_k = N^{-1}M(H_M - H_{M-N})$. Эти значения приблизительно равны значениям при равномерном исследовании, но несколько выше за счет возможности повторного опробования одного и того же места. В действительности при $4 = N < M \le 16$ линейное исследование имеет лучшие характеристики!

На практике нельзя применять бесконечно много хеш-функций; в качестве последнего средства должна использоваться другая схема — схема наподобие линейного исследования. Этот метод хуже описанных в тексте и представляет только историческую ценность, так как он послужил толчком к разработке метода Морриса (Morris), который, в свою очередь, привел к разработке алгоритма D. См. *CACM* **6** (1963), 101, где М. Д. Мак-Илрой (M. D. McIlroy) приписывает эту идею В. А. Высоцки (V. A. Vyssotsky); та же технология была открыта в 1956 году А. В. Хольтом (A. W. Holt), который успешно применил ее в системе GPX для UNIVAC.

49. $C'_N - 1 = \sum_{k>b}(k-b)P_{Nk} \approx \sum_{k>b}(k-b)e^{-\alpha b}(\alpha b)^k/k! = \alpha b t_b(\alpha)$. [*Примечание.* В общем случае имеем

$$\sum_{b\ge 0}\Bigl(\sum_{k>b}(k-b)P_k\Bigr)z^b = \frac{P'(1)}{1-z} + \frac{z(P(z)-1)}{(1-z)^2}$$

для любой производящей функции вероятностей $P(z) = P_0 + P_1 z + \cdots$.]

$$\begin{aligned} C_N - 1 &= \frac{M}{N}\sum_{k>b}\binom{k-b+1}{2}P_{Nk} \\ &= \frac{M}{2N}\sum_{k>b}(k(k-1) - 2k(b-1) + b(b-1))P_{Nk} \\ &= \tfrac{1}{2}e^{-b\alpha}(b\alpha)^b b!^{-1}(b + b\alpha - 2b + 2 + (b\alpha^2 - 2\alpha(b-1) + b - 1)R(\alpha, b)). \end{aligned}$$

[В 1957 году анализ успешного поиска с цепочками был впервые проведен В. П. Хайзингом (W. P. Heising). Простые выражения в (57) и (58) были найдены Я. А. ван дер Пулом (J. A. van der Pool) в 1971 году; им также рассмотрен вопрос о минимизации функции, представляющей комбинированную стоимость используемого пространства и количества обращений. Можно определить дисперсию C'_N и числа переполнений на блок, исходя из $\sum_{k>b}(k-b)^2 P_{Nk} = (2N/M)(C_N - 1) - (C'_N - 1)$. Дисперсия общего количества переполнений может быть приближенно представлена увеличенной в M раз дисперсией в единичном блоке, хотя на самом деле это завышенное значение, потому что общее количество записей вынуждено быть равным N. Истинная дисперсия может быть найдена, как и в упр. 37. Обратитесь также к исследованию χ^2-критерия в разделе 3.3.1C.]

50. А затем — что $Q_0(M, N-1) = (M/N)(Q_0(M,N) - 1)$. В общем случае $rQ_r(M,N) = MQ_{r-2}(M,N) - (M-N-r)Q_{r-1}(M,N) = M(Q_{r-1}(M,N+1) - Q_{r-1}(M,N))$; $Q_r(M,N-1) = (M/N)(Q_r(M,N) - Q_{r-1}(M,N))$.

51. $R(\alpha, n) = \alpha^{-1}(n!\, e^{\alpha n}(\alpha n)^{-n} - Q_0(\alpha n, n))$.

52. См. формулу 1.2.11.3–(9) и упр. 3.1–14.

53. Согласно 1.2.11.3–(8) получаем $\alpha(\alpha n)^n R(\alpha, n) = e^{\alpha n}\gamma(n+1, \alpha n)$; следовательно, в соответствии с указанным упражнением $R(\alpha, n) = (1-\alpha)^{-1} - (1-\alpha)^{-3}n^{-1} + O(n^{-2})$. [Эта асимптотическая формула может быть получена непосредственно, с использованием метода (43), если заметить, что коэффициент при α^k в $R(\alpha, n)$ равен

$$1 - \binom{k+2}{2}n^{-1} + O(k^4 n^{-2}).$$

В действительности согласно 1.2.9–(28) коэффициент при α^k равен

$$\sum_{r\geq 0}(-1)^r n^{-r}\left\{ {r+k+1 \atop k+1} \right\}.]$$

54. Используя указание, а также формулы 1.2.6–(53) и 1.2.6–(49), получим

$$\sum_{b\geq 1} t_b(\alpha) = \sum_{m\geq 1}\frac{\alpha^m}{(m+1)(m)m!}\sum_k \binom{m}{k}(-1)^{m-k}k^{m+1} = \sum_{m\geq 1}\alpha^m/2.$$

Указание следует из хорошо известного гипергеометрического тождества Куммера

$$e^{-z}F(a; b; z) = F(b-a; b; -z),$$

поскольку $(n+1)!\,t_n(\alpha) = e^{-n\alpha}(\alpha n)^n F(2; n+2; \alpha n)$; см. *Crelle* **15** (1836), 39–83, 127–172, формула 26.4.

55. Если $B(z)C(z) = \sum s_i z^i$, имеем $c_0 = s_0 + \cdots + s_b$, $c_1 = s_{b+1}$, $c_2 = s_{b+2}, \ldots$; следовательно, $B(z)C(z) = z^b C(z) + Q(z)$. $P(z) = z^b$ имеет $b-1$ корень q_j, где $|q_j| < 1$, определяемые как решения уравнения $e^{\alpha(q_j-1)} = \omega^{-j}q_j$, $\omega = e^{2\pi i/b}$. Для решения $e^{\alpha(q-1)} = \omega^{-1}q$ положим $t = \alpha q$ и $z = \alpha\omega e^{-\alpha}$, так что $t = ze^t$. Согласно формуле Лагранжа получим

$$\begin{aligned}\frac{1}{1-q} &= 1 + \sum_{r\geq 0} r \sum_{n\geq r}\frac{n^{n-r-1}\omega^n\alpha^{n-r}e^{-n\alpha}}{(n-r)!}\\ &= 1 + \sum_{r\geq 1} r \sum_{m\geq 0}\frac{\alpha^m}{m!}(-1)^m \sum_{n\geq r}\binom{m}{n-r}(-1)^{n-r}\omega^n n^{m-1}.\end{aligned}$$

По теореме Абеля о возможности перехода к пределу для степенного ряда, устремляя $|\omega| \to 1$ так, что ω находится внутри единичного круга, формулу можно привести к виду

$$\frac{1-\alpha\omega}{1-\omega} + \sum_{m\geq 2}\frac{\alpha^m}{m!}(-1)^m \sum_{n\geq 0}\binom{m-2}{n}(-1)^n\omega^{n+1}(n+1)^{m-1}.$$

Теперь, заменяя ω на ω^j и суммируя по $1 \leq j < b$, получаем

$$\frac{b-1}{2} + \alpha\frac{b-1}{2} + \sum_{m\geq 2}\alpha^m\left(-\frac{1}{2} + \frac{(-1)^m}{m!}b\sum_{n\geq 1}\binom{m-2}{nb-1}(-1)^{nb-1}(nb)^{m-1}\right),$$

откуда после небольших манипуляций с использованием указания к упр. 54 получается требуемый результат.

Начало этому анализу, применяемому при решении задач различных типов, было положено в работах N. T. J. Bailey, *J. Roy. Stat. Soc.* **B16** (1954), 80–87; M. Tainiter, *JACM* **10** (1963), 307–315; A. G. Konheim and B. Meister, *JACM* **19** (1972), 92–108.

56. См. Blake and Konheim, *JACM* **24** (1977), 591–606. Альфредо Виола (Alfredo Viola) и Патрицио Поблете (Patricio Poblete) [*Algorithmica* **21** (1998), 37–71] показали, что

$$C_{Mb} = 1 + \frac{M-1}{2Mb} + \frac{1}{b}\sum_{j\ge 2}\binom{bm-1}{j}m^{-j}\sum_{k\ge 1}\binom{j-2}{bk-1}(-1)^{j+bk-1}k^{j-1}$$
$$= \sqrt{\frac{\pi M}{8b}} + \frac{1}{3b} + \frac{1}{b}\sum_{j=1}^{b-1}\frac{1}{(1-T(e^{2\pi ij/b-1}))} + \frac{1}{24}\sqrt{\frac{\pi}{2b^3M}} + O(b^{-2}M^{-1}),$$

где T — функция дерева из 2.3.4.4–(30).

58. 0 1 2 3 4 и 0 2 4 1 3 плюс аддитивные сдвиги 1 1 1 1 1 mod 5, каждый с вероятностью $\frac{1}{10}$. Аналогично при $M = 6$ требуется 30 перестановок, и решение существует, начинаясь с

$$\tfrac{1}{20}\times 012345,\quad \tfrac{1}{60}\times 013254,\quad \tfrac{1}{60}\times 024315,\quad \tfrac{1}{20}\times 023451,\quad \tfrac{1}{30}\times 034125.$$

При $M = 7$ требуется 49 перестановок, и решение порождается из

$$\tfrac{1}{35}\times 0123456,\ \tfrac{2}{105}\times 0153246,\ \tfrac{1}{35}\times 0243516,\ \tfrac{2}{105}\times 0263145,$$
$$\tfrac{1}{35}\times 0361425,\ \tfrac{1}{105}\times 0326415,\ \tfrac{1}{105}\times 0315426.$$

59. Ни одна перестановка не может иметь бóльшую вероятность, чем $1/\binom{M}{\lfloor M/2\rfloor}$, поэтому должно быть не менее $\binom{M}{\lfloor M/2\rfloor} = \exp(M\ln 2 + O(\log M))$ перестановок с ненулевыми вероятностями.

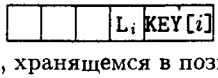
60. Предварительные результаты получены в работе Ajtai, Komlós, and Szemerédi, *Information Processing Letters* **7** (1978), 270–273.

62. См. дискуссию в *AMM* **81** (1974), 323–343, где представлены наилучшие циклические последовательности хеширования для $M \le 9$.

63. MH_M согласно упр. 3.3.2–8; стандартное отклонение составляет $\approx \pi M/\sqrt{6}$.

64. Среднее количество перемещений составляет $\frac{1}{2}(N-1)/M + \frac{2}{3}(N-1)(N-2)/M^2 + \frac{3}{4}(N-1)(N-2)(N-3)/M^3 + \cdots \approx \frac{1}{1-\alpha} - \frac{1}{\alpha}\ln\frac{1}{1-\alpha}$. [Аналогичная задача решена в *Comp. J.* **17** (1974), 139–140.]

65. Ключи могут храниться в отдельной таблице с последовательной организацией (предположим, что все удаления, если они производятся, соответствуют стековому представлению LIFO (last-in-first-out, "последним вставляется — первым удаляется"). Элементами хеш-таблицы являются указатели на эту "таблицу имен"; например, TABLE[i] может иметь вид



где L$_i$ — количество слов в ключе, хранящемся в позициях KEY[i], KEY[i] + 1, ...

Остаток элементов хеш-таблицы может использоваться несколькими способами: (i) как ссылка для алгоритма C; (ii) как часть информации, связанной с ключом; (iii) как "вторичный хеш-код". Последняя идея, предложенная Робертом Моррисом (Robert Morris), иногда позволяет ускорить поиск (мы рассматриваем ключ KEY[i] только в том случае, когда значение некоторой функции $h_2(K)$ соответствует вторичному хеш-коду).

66. Да; при этом расположить записи можно лишь одним способом. Среднее количество проб при неудачном поиске снижается до C_{N-1}, хотя и остается равным C'_N при вставке N-го элемента. Эта важная технология именуется *упорядоченным хешированием*. См. *Comp. J.* **17** (1974), 135–142; D. E. Knuth, *Literate Programming* (1992), 144–149, 216–217.

67. (a) Если в (44) $c_j = 0$, оптимального расположения можно достичь, рассортировав a согласно невозрастающему "циклическому порядку", которым предполагается, что $j-1 > \cdots > 0 > M-1 > \cdots > j$. (b) Между шагами L2 и L3 поменяйте вставляемую запись и

TABLE[i], если последняя ближе к начальному положению, чем предыдущая. [Этот алгоритм, названный хешированием Робин Гуда (Robin Hood hashing) в работе Celis, Larson, and Munro, *FOCS* **26** (1985), 281–288, представляет собой вариант упорядоченного хеширования.] (c) Обозначим через $h(m,n,d)$ количество хеш-последовательностей, которые приводят к $c_0 \le d$. Можно показать [*Comp. J.* **17** (1974), 141], что $(h(m,n,d) - h(m,n,d-1))M$ — общее количество перемещений $d > 0$ по всем M^N хеш-последовательностям и что можно записать $h(M,N,d) = a(M,N,d+1) - Na(M,N-1,d+1)$, где $a(m,n,d) = \sum_{k=0}^{d} \binom{n}{k}(m+d-k)^{n-k}(k-d)^k$. Сложные вычисления с использованием методов из упр. 28 и 50 показывают, что среднее значение $\sum d_j^2$ равно

$$\begin{aligned} M^{1-N}\sum_{d=1}^{N} d^2(h(M,N,d) - h(M,N,d-1)) \\ &= \frac{M^2}{2} + \frac{2M}{3} + \frac{N}{6} + \frac{N^2}{6M} - \frac{N}{6M} - M\Big(\frac{M}{2} - \frac{N}{2} + \frac{2}{3}\Big)Q_0(M,N) \\ &= M\left(\frac{1}{2(1-\alpha)^2} - \frac{7}{6(1-\alpha)} + \frac{2}{3} + \frac{\alpha}{6} + \frac{\alpha^2}{6}\right) + O(1) \end{aligned}$$

при $N = \alpha M$. Если алгоритм не модифицирован (см. упр. 28), $\mathrm{E}\sum d_j^2$ преобразуется в

$$\begin{aligned} \frac{M}{3}(Q_2(M,N) - Q_1(M,N)) - \frac{M}{2}(Q_0(M,N) - 1) + \frac{N}{6} \\ = M\left(\frac{1}{3(1-\alpha)^3} - \frac{1}{3(1-\alpha)^2} - \frac{1}{2(1-\alpha)} + \frac{1}{2} + \frac{\alpha}{6}\right) + O(1). \end{aligned}$$

Если все записи имеют приблизительно одинаковое перемещение d и если успешный поиск осуществляется значительно чаще, чем неудачный, то выгодно начинать с позиции $h' = h(K) + d$, а затем опробовать позиции $h'-1$, $h'+1$, $h'-2$ и т. д. П. В. Поблете (P. V. Poblete), А. Виола (A. Viola) и Д. Я. Мунро (J. I. Munro) показали [*Random Structures and Algorithms* **10** (1997), 221–255], что $\sum d_j^2$ может быть сделана столь же малой, как и в методе Робин Гуда, при помощи более простого подхода, называемого хешированием "последним пришел — первым обслужен" (last-come-first-served), при котором каждый вновь вставляемый ключ помещается в свою начальную позицию; все другие ключи перемещаются на один шаг, пока не будет найдено пустое место. Оба эти подхода применимы как к двойному хешированию, так и к линейному исследованию, но уменьшение количества проб не компенсирует увеличения времени, требуемого для одной пробы с учетом двойного хеширования, пока таблица не заполнится почти до отказа (см. Poblete and Munro, *J. Algorithms* **10** (1989), 228–248).

68. Среднее значение $(d_1 + \cdots + d_N)^2$ равно

$$\begin{aligned} \frac{N}{12}((M-N)^3 + (N+3)(M-N)^2 + (8N+1)(M-N) + 5N^2 + 4N - 1 \\ - ((M-N)^3 + 4(M-N)^2 + (6N+3)(M-N) + 8N)Q_0(M,N-1)), \end{aligned}$$

как можно показать, рассмотрев связь между задачей о парковке и связанными графами, о которой упоминалось в упр. 31. Для получения дисперсии среднего количества проб в случае успешного поиска разделим эту величину на N^2 и вычтем $\frac{1}{4}(Q_0(M,N-1) - 1)^2$; асимптотически это равно $\frac{1}{12}((1+2\alpha)/(1-\alpha)^4 - 1)/N + O(N^{-2})$. (См. P. Flajolet, P. V. Poblete, and A. Viola, *Algorithmica* **22** (1998), 490–515; D. E. Knuth, *Algorithmica* **22** (1998), 561–568. Вычисленная дисперсия должна отличаться от общей дисперсии, которая равна $\mathrm{E}\sum d_j^2/N - \frac{1}{4}(Q_0(M,N-1) - 1)^2$; см. ответы к упр. 37 и 67.)

69. Пусть $q_k = p_k + p_{k+1} + \cdots$; тогда неравенство $q_k \geq \max(0, 1-(k-1)(M-n)/M)$ дает нижнюю границу $C'_N = \sum_{k\geq 1} q_k$.

70. Люкер (Lueker) и Молодович (Molodowitch) [*Combinatorica* **13** (1993), 83–96] привели замечательно простое доказательство подобного результата, но у них появляется дополнительный множитель $(\log M)^2$ под знаком O; указанный результат получается тем же путем с помощью определенных трюков при оценке вероятностей. А. Р. Сигель (A. R. Siegel) и Ж. П. Шмидт (J. P. Schmidt) показали, что в действительности ожидаемое количество проб при двойном хешировании составляет $1/(1-\alpha) + O(1/M)$ для фиксированного $\alpha = N/M$. [Computer Science Tech. Report 687 (New York: Courant Institute, 1995).]

72. [*J. Comp. Syst. Sci.* **18** (1979), 143–154.] (a) При данных ключах $K_1, \ldots, K_N$ и K вероятность того, что K_j находится в том же списке, что и K, равна $\leq 1/M$, если $K \neq K_j$. Следовательно, ожидаемый размер списка равен $\leq 1 + (N-1)/M$.

(b) Предположим, что существует Q возможных символов и имеется M^Q возможных вариантов выбора для каждого h_j. Случайный выбор каждого h_j эквивалентен выбору случайной строки из матрицы H из M^{Ql} строк и Q^l столбцов с элементом $h(x_1 \ldots x_l) = (h_1(x_1) + \cdots + h_l(x_l)) \bmod M$ в столбце $x_1 \ldots x_l$. В столбцах $K = x_1 \ldots x_l$ и $K' = x'_1 \ldots x'_l$ с $x_j \neq x'_j$ для некоторого j имеем $h(K) = (s + h_j(x_j)) \bmod M$ и $h(K') = (s' + h_j(x'_j)) \bmod M$, где $s = \sum_{i\neq j} h_i(x_i)$ и $s' = \sum_{i\neq j} h_i(x'_i)$ независимы от h_j. Значения $h_j(x_j) - h_j(x'_j)$ равномерно распределены по модулю M; следовательно, мы имеем $h(K) = h(K')$ с вероятностью $1/M$ без учета значений s и s'.

(c) Да; добавление любой константы к $h_j(x_j)$ прибавляет к $h(x)$ константу по модулю M.

73. (i) Это специальный вариант упражнения 72, (c), в котором каждый ключ рассматривается как последовательность битов, а не символов. (ii) Из доказательства (b) следует, что достаточно показать равномерность распределения $h_j(x_j) - h_j(x'_j)$ по модулю M при $x_j \neq x'_j$. В действительности вероятность того, что $h_j(x_j) = y$ и $h_j(x'_j) = y'$, равна $1/M^2$ для любых данных y и y', поскольку уравнения $a_j x_j + b_j \equiv y$ и $a_j x'_j + b_j \equiv y'$ по модулю с простым M имеют единственное решение (a_j, b_j) для любых данных (y, y').

Если M не является простым и p — простое число, большее, чем M, подобный результат имеет место, если предположить $h_j(x_j) = ((a_j x_j + b_j) \bmod p) \bmod M$, где a_j и b_j выбраны случайным образом по модулю p. В этом случае семейство не является полностью универсальным, но достаточно близкó к таковому для достижения практических целей: вероятность коллизии различных ключей не превышает $1/M + r(M-r)/Mp^2 \leq 1/M + M/4p^2$, где $r = p \bmod M$.

74. В целом, это утверждение ложно. Например, предположим, что $M = N = n^2$, и рассмотрим матрицу H с $\binom{N}{n}$ строками, по одной для каждого варианта размещения n нулей в различных столбцах; ненулевыми являются элементы $1, 2, \ldots, N-n$ слева направо в каждой строке. Эта матрица универсальна, потому что в каждой паре столбцов имеется $\binom{N-2}{n-2} = \binom{N}{n}\frac{n}{N}\frac{n-1}{N-1} < \binom{N}{n}\left(\frac{n}{N}\right)^2 = R/M$ совпадений; однако число нулей в каждой строке составляет $\sqrt{N} \neq O(1) + O(N/M)$.

Примечание. Этот пример указывает, что ожидаемый размер списка существенно отличается от ожидаемого количества коллизий при вставке нового ключа. Рассмотрим предположение $h(x_1 \ldots x_l) = h_1(x_1)$, где h_1 выбрано случайным образом. Такое семейство хеш-функций делает ожидаемый размер каждого списка равным N/M; конечно, это еще не универсальное семейство, потому что множество из N ключей, имеющих один и тот же первый символ x_1, приведет к образованию одного списка размером N и пустых прочих списков. Ожидаемое количество коллизий будет равно $N(N-1)/2$, однако для универсального хеш-семейства это число не превышает $N(N-1)/2M$ независимо от множества ключей.

С другой стороны, *можно* показать, что ожидаемый размер каждого списка в универсальном семействе равен $O(1)+O(N/\sqrt{M})$. Предположим, что в строке h имеется z_h нулей. Тогда в ней содержится как минимум $\binom{z_h}{2}$ пар одинаковых элементов. Максимум $\sum_{h=1}^{R} x_h$ при условии $\sum_{h=1}^{R}\binom{z_h}{2} \le \binom{N}{2}R/M$ достигается, когда каждое z_h равно z, где $\binom{z}{2} = \binom{N}{2}/M$, а именно

$$z = \frac{1}{2} + \sqrt{\frac{1}{4} + \frac{N(N-1)}{M}} < 1 + \sqrt{\frac{N(N-1)}{M}}.$$

75. (a) Очевидно, что утверждение истинно, даже если $h_2, \ldots, h_l$ тождественно равны нулю. (b) Верно в соответствии с ответом к 72, (b). (c) Верно. Результат ясен, если все K, K' и K'' отличаются в некоторой позиции символа. В противном случае примем, что $x_j = x'_j \ne x''_j$ и $x_k \ne x'_k = x''_k$. Тогда величины $h_j(x_j) + h_k(x_k)$, $h_j(x_j) + h_k(x'_k)$ и $h_j(x''_j) + h_k(x'_k)$ независимы одна от другой, равномерно распределены и не зависят от других $l-2$ символов ключей. (d) Неверно. Рассмотрим, например, случай для $M = l = 2$ с однобитовыми символами. Тогда все четыре ключа хешируются в одну и ту же позицию с вероятностью 1/4.

76. Используем $h(K) = (h_0(l) + h_1(x_1) + \cdots + h_l(x_l)) \bmod M$, где каждая функция h_j выбирается, как в упр. 73. Случайные коэффициенты для h_j (и при желании предварительно вычисленный массив значений) генерируются при первом появлении ключа длиной $\ge j$. Поскольку l не ограничено, матрица H бесконечна; однако в реальной работе программы будет использоваться только ее конечная часть.

77. Пусть $p \le 2^{-16}$ — вероятность того, что два 32-битовых ключа имеют один и тот же образ в H. Наихудшая ситуация складывается, когда два данных ключа совпадают в семи из их восьми 32-битовых подключей; значит, вероятность коллизии равна $1-(1-p)^4 < 4p$. [См. Wegman and Carter, *J. Comp. Syst. Sci.* **22** (1981), 265–279.]

РАЗДЕЛ 6.5

1. Путь, описанный в указании, может быть преобразован посредством замены каждого идущего вниз шага из точки $(i-1, j)$ к "новому рекордно низкому" значению $(i, j-1)$ шагом, идущим вверх. Если предпринимается c таких изменений, путь заканчивается в $(m, n-2t+2c)$, где $c \ge 0$ и $c \ge 2t-n$; следовательно, $n-2t+2c \ge n-2k$. В перестановке, соответствующей измененному пути, наименьшие c элементов списка B отвечают измененным шагам вниз, а в списке A содержится $t-c$ элементов, соответствующих неизмененным шагам вниз.

Нетрудно увидеть, что при $t = k$ построение обратимо; следовательно, строится в точности $\binom{n}{k}$ перестановок. Заметим, что в соответствии с этим доказательством содержимое списков A и C может располагаться в произвольном порядке.

Примечание. Мы сосчитали эти пути в упр. 2.2.1–4 несколько иным способом. При $k = \lfloor n/2 \rfloor$ данное построение доказывает *лемму Спернера*, которая гласит, что невозможно иметь более $\binom{n}{\lfloor n/2 \rfloor}$ подмножеств множества $\{1, 2, \ldots, n\}$, таких, что ни одно подмножество не содержится в другом. [Эмануэль Спернер (Emanuel Sperner), *Math. Zeitschrift* **27** (1928), 544–548.] Если бы существовал такой набор подмножеств, каждая из $\binom{n}{k}$ перестановок могла бы иметь не более одного из подмножеств в начальных позициях; в то же время каждое подмножество появляется в некоторой перестановке. Построение, использованное здесь, представляет скрытую форму более общей конструкции, с помощью которой Н. Г. де Брейн (N. G. de Bruijn), К. ван Эббенхорст Тенгберген (C. van Ebbenhorst Tengbergen) и Д. Круйсвик (D. Kruyswijk) [*Nieuw Archief voor Wiskunde* (2) **23** (1951), 191–193] доказали обобщение леммы Спернера для мультимножеств: "Пусть M — мультимножество, содержащее n элементов (считая повторения). Набор всех $\lfloor n/2 \rfloor$-элементных мультиподмножеств M представляет собой наибольший возможный набор, такой, что ни одно мультиподмножество не содержится в другом". Например, наибольший такой набор

при $M = \{a,a,b,b,c,c\}$ состоит из семи мультиподмножеств: $\{a,a,b\}$, $\{a,a,c\}$, $\{a,b,b\}$, $\{a,b,c\}$, $\{a,c,c\}$, $\{b,b,c\}$, $\{b,c,c\}$. Этот набор соответствует семи перестановкам шести атрибутов (A_1, B_1, A_2, B_2, A_3, B_3) для случая, когда запрос, включающий A_i, содержит также B_i. Дополнительные комментарии по этому вопросу приводятся в статье C. Greene and D. J. Kleitman, *J. Combinatorial Theory* **A20** (1976), 80–88.

2. Пусть a_{ijk} — список всех ссылок на записи, имеющие значения трех атрибутов (i,j,k), и предположим, что список a_{011} — самый короткий среди списков a_{011}, a_{101}, a_{110}. Тогда списком с минимальной длиной является $a_{001}a_{011}a_{111}a_{101}a_{100}a_{110}a_{111}a_{011}a_{010}$. Однако, если a_{011} пуст и пуст также любой из списков a_{001}, a_{010} и a_{100}, длина может быть сокращена путем удаления одного из двух вхождений a_{111} [*CACM* **15** (1972), 802–808].

3. (a) Зерна аниса и/или мед, возможно, в комбинации с мускатным орехом и/или ванилином. (b) Никакие.

4. Пусть p_t — вероятность того, что запрос включает в точности t-битовые позиции, и пусть P_t — вероятность того, что t данных позиций в случайной записи равны 1. Тогда ответом будет $\sum_t p_t P_t$ минус вероятность того, что искомая запись действительно найдена; последняя вероятность равна $\binom{N-q}{r-q}/\binom{N}{r}$, где $N = \binom{n}{k}$. По принципу включения и исключения

$$P_t = \sum_{j\ge 0}(-1)^j\binom{t}{j}f(n-j,k,r)/f(n,k,r),$$

где $f(n,k,r)$ — количество возможных выборов r различных k-битовых кодов атрибутов в n-битовых полях, а именно

$$f(n,k,r) = \binom{\binom{n}{k}}{r}.$$

При $q = r$ имеем согласно упр. 1.3.3–26

$$p_t = \sum_{l\ge 0}(-1)^l\binom{t+l}{t}\binom{n}{t+l}P_{t+l} = \binom{n}{t}\sum_{j\ge 0}(-1)^j\binom{t}{j}f(t-j,k,q)/f(n,k,q).$$

Примечание. Приведенные выше вычисления впервые были выполнены в более общей форме в G. Orosz and L. Takács, *J. of Documentation* **12** (1956), 231–234. Легко показать, что среднее значение $\sum tp_t$ равно $n(1-f(n-1,k,q)/f(n,k,q))$. Предположение о том, что коды случайных атрибутов в записях и запросах необязательно должны быть различными, как в случаях использования технологий Харрисона и Блюма, может быть проанализировано таким же образом с $f(n,k,r) = \binom{n}{k}^r$. Когда параметры находятся в соответствующих диапазонах, получаем $P_t \approx (1-e^{-kr/n})^t$ и $\sum p_t P_t \approx P_{n(1-\exp(-kq/n))}$.

6. $L(t) = \sum_j \binom{m_1}{j}\binom{m_2}{t-j}L_1(j)L_2(t-j)/\binom{m_1+m_2}{t}$ (следовательно, если $L_1(t) \approx N_1\alpha^{-t}$ и $L_2(t) \approx N_2\alpha^{-t}$, то $L(t) \approx N_1N_2\alpha^{-t}$).

7. (a) $L(1) = 3$, $L(2) = 1\frac{3}{4}$. (b) $L(1) = 3\frac{3}{4}$, $L(2) = 2\frac{1}{3}$, $L(3) = 1\frac{9}{16}$. [*Примечание.* Тривиальное проецирующее отображение типа $00{*}{*} \to 0$, $01{*}{*} \to 1$, $10{*}{*} \to 2$, $11{*}{*} \to 3$ представляет собой отображение, наихудший случай которого хуже, но средний — лучше: $L(1) = 3$, $L(2) = 2\frac{1}{6}$, $L(3) = 1\frac{1}{2}$.]

8. (a) При $S = S_0 0 \cup S_1 1$ имеем $f_t(S) = f_t(S_0 \cup S_1) + f_{t-1}(S_0) + f_{t-1}(S_1)$. Таким образом, $f_t(s,m)$ представляет собой минимум $f_t(s_0,m-1) + f_{t-1}(s_0,m-1) + f_{t-1}(s_1,m-1)$ по всем s_0 и s_1, таким, что $2^{m-1} \ge s_0 \ge s_1 \ge 0$ и $s_0 + s_1 = s$. Чтобы доказать, что минимум достигается для $s_0 = \lceil s/2\rceil$ $s_1 = \lfloor s/2\rfloor$, можно использовать индукцию по m с очевидным результатом при $m = 1$: дано $m \ge 2$. Пусть $g_t(s) = f_t(s,m-1)$ и $h_t(s) = f_t(s,m-2)$. Тогда по индукции $g_t(s_0) + g_{t-1}(s_0) + g_{t-1}(s_1) = h_t(\lceil s_0/2\rceil) + h_{t-1}(\lceil s_0/2\rceil) + h_{t-1}(\lfloor s_0/2\rfloor) + h_{t-1}(\lceil s_0/2\rceil) + h_{t-2}(\lceil s_0/2\rceil) + h_{t-2}(\lfloor s_0/2\rfloor) + h_{t-1}(\lceil s_1/2\rceil) + h_{t-2}(\lceil s_1/2\rceil) + h_{t-2}(\lfloor s_1/2\rfloor)$, что $\ge g_t(\lceil s_0/2\rceil + \lceil s_1/2\rceil) + g_{t-1}(\lceil s_0/2\rceil + \lceil s_1/2\rceil) + g_{t-1}(\lfloor s_0/2\rfloor + \lfloor s_1/2\rfloor)$. Если $s_0 > s_1 + 1$, имеем

$\lceil s_0/2\rceil + \lceil s_1/2\rceil < s_0$, за исключением случая, когда $s_0 = 2k+1$ и $s_1 = 2k-1$. В последнем случае, однако, $g_t(s_0) + g_{t-1}(s_0) + g_{t-1}(s_1) \geq h_t(2k+1) + 2h_{t-1}(2k) \geq h_t(2k) + 2h_{t-1}(2k)$.

(b) Заметьте, что множество S, содержащее числа 0, 1, ..., $s-1$ в двоичной записи, имеет такое свойство: $S_0 \cup S_1 = S_0$ и S_0 содержит $\lceil s_0/2\rceil$ элементов. Отсюда следует, что $f_t(2^{m-n}, m) = [z^t]\,(1+z)^n(1+2z)^{m-n}$.

10. (a) Должно существовать $\frac{1}{6}v(v-1)$ троек, и x_v должно встречаться в $\frac{1}{2}v$ из них. (b) Поскольку v нечетно, для каждого i имеется уникальная тройка $\{x_i, y_j, z\}$, откуда легко показать, что S' — Штейнеровская система троек. Отсутствующие в K' пары — $\{z, x_2\}$, $\{x_2, y_2\}$, $\{y_2, x_3\}$, $\{x_3, y_3\}$, ..., $\{x_{v-1}, y_{v-1}\}$, $\{y_{v-1}, x_v\}$, $\{x_v, z\}$. (d) Начав со случая для $v = 1$ и применив операции $v \to 2v-2$, $v \to 2v+1$, получим все неотрицательные числа, которые не имеют вид $3k+2$, потому что случаи $6k + (0,1,3,4)$ получаются из меньших чисел $3k + (1,0,1,3)$ соответственно.

Оказывается, "Штейнеровские системы троек" не должны именоваться Штейнеровскими, хотя это имя глубоко укоренилось в литературе. Публикация Штейнера [*Crelle* **45** (1853), 181–182] появилась несколькими годами позже публикации Киркмана. Феликс Клейн (Felix Klein) заметил [*Vorlesungen über die Entwicklung der Math. im 19. Jahrhundert* **1** (Springer, 1926), 128], что в последние годы жизни Штейнер цитировал английских авторов без ссылок на них. Более того, концепция троек появилась еще раньше в двух хорошо известных книгах Ю. Плюкера (J. Plücker) [*System der analytischen Geometrie* (1835), 283–284; *Theorie der algebraischen Curven* (1839), 245–247].

11. Возьмем Штейнеровскую систему троек над $2v+1$ объектами. Назовем один из объектов z, а другие объекты переименуем таким образом, чтобы тройками, содержащими z, оказались $\{z, x_i, \bar{x}_i\}$. Затем удалим эти тройки.

12. $\{k, (k+1) \bmod 14, (k+4) \bmod 14, (k+6) \bmod 14\}$ при $0 \leq k < 14$, где $(k+7) \bmod 14$ представляет собой дополнение k (комплементарная система является частным случаем *групп, разделяемых на блоки*; см. Bose, Shrikhande, and Bhattacharya, *Ann. Math. Statistics* **24** (1953), 167–195).

14. Самый простой вид удаления — из k-d-деревьев (замещение для корня может быть найдено примерно за $O(N^{1-1/k})$ шагов). В четревьях для удаления, похоже, потребуется полная перестройка поддерева, корнем которого является удаляемый узел (однако такое поддерево содержит в среднем лишь около $\log N$ узлов). Для почтовых деревьев удаление практически безнадежно.

16. Пусть каждая тройка соответствует кодовому слову и каждое кодовое слово имеет ровно три равных единице бита, определяющих элементы соответствующей тройки. Если u, v и w — различные кодовые слова, то слово u имеет не более двух равных единице битов с суперпозицией v и w, поскольку оно может иметь не более одного общего бита со словами v и w по отдельности (аналогично из системы четверок порядка v можно построить $v(v-1)/12$ кодовых слов, никакое из которых не содержится в суперпозиции любых трех других, и т. д.).

17. (a) Пусть $c_0 = b_0$, а для $1 \leq k \leq n$ положим $c_k =$ (если $b_{k-1} = 0$, то "*", иначе — b_k), $c_{-k} =$ (если $b_{k-1} = 1$, то "*", иначе — b_k). Тогда базовый запрос $c_{-n} \ldots c_0 \ldots c_n$ описывает содержимое блока $b_0 \ldots b_n$. (Следовательно, эта схема представляет собой частный случай комбинаторного хеширования и ее среднее время запроса соответствует нижней границе в упр. 8, (b).)

(b) Пусть $d_k =$ [бит k определен] для $-n \leq k \leq n$. Можно положить, что $d_{-k} \leq d_k$ при $1 \leq k \leq n$. Тогда максимальное количество проверенных блоков получается, когда все определенные биты равны нулю и оно может быть вычислено следующим образом.

Установим $x \leftarrow y \leftarrow 1$; затем для $k = n, n-1, \ldots, 0$ выполним $(x,y) \leftarrow (x,y)M_{d_{-k}+d_k}$, где

$$M_0 = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}, \qquad M_1 = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}, \qquad M_2 = \begin{pmatrix} 1 & 1 \\ 0 & 0 \end{pmatrix}.$$

И, наконец, выведем x (который может равняться y после $k = 0$).

Будем говорить, что $(x,y) \succeq (x',y')$, если $x \geq x'$ и $x + y \geq x' + y'$. Тогда, если $(x,y) \succeq (x',y')$, имеем $(x,y)M_d \succeq (x',y')M_d$ для $d = 0, 1, 2$. Теперь

$$\begin{aligned}
(x,y)M_2M_1^jM_0 &= (F_{j+3}x, F_{j+3}x), \\
(x,y)M_1M_1^jM_1 &= (F_{j+3}x + F_{j+2}y, F_{j+2}x + F_{j+1}y), \\
(x,y)M_0M_1^jM_2 &= (F_{j+2}x + F_{j+2}y, F_{j+2}x + F_{j+2}y);
\end{aligned}$$

следовательно, $(x,y)M_1M_1^jM_1 \succeq (x,y)M_2M_1^jM_0$, потому что $2y \geq x$; и аналогично $(x,y)M_1M_1^jM_1 \succeq (x,y)M_0M_1^jM_2$, потому что $x \geq y$. Отсюда вытекает, что наихудший случай возникает либо при $d_{-k} + d_k \leq 1$ для $1 \leq k \leq n$, либо при $d_{-k} + d_k \geq 1$ для $1 \leq k \leq n$. Кроме того,

$$\begin{aligned}
(x,y)M_0M_1^j &= (F_{j+2}x + F_{j+2}y, F_{j+1}x + F_{j+1}y), \\
(x,y)M_1^jM_0 &= (F_{j+2}x + F_{j+1}y, F_{j+2}x + F_{j+1}y), \\
(x,y)M_2M_1^j &= (F_{j+2}x, F_{j+1}x), \\
(x,y)M_1^jM_2 &= (F_{j+1}x + F_jy, F_{j+1}x + F_jy).
\end{aligned}$$

Значит, в худшем случае требуется следующее количество блоков:

$$\begin{array}{lll}
2^{n-t}F_{t+3}, & \text{если } 0 \leq t \leq n & [\text{из } M_1^tM_0^{n+1-t}]; \\
2^{t-n}F_{3n-2t+3}, & \text{если } n \leq t \leq \lceil 3n/2 \rceil & [\text{из } M_1^{3n-2t}(M_1M_2)^{t-n}M_0]; \\
2^{2n+1-t}, & \text{если } \lceil 3n/2 \rceil \leq t \leq 2n & [\text{из } M_2^{2t-3n}(M_1M_2)^{2n-t}M_0].
\end{array}$$

[Эти результаты, по сути, были получены В. О. Буркхардом (W. A. Burkhard), *BIT* **16** (1976), 13–31, и обобщены в *J. Comp. Syst. Sci.* **15** (1977), 280–299; более сложное отображение Буркхарда из $a_0 \ldots a_{2n}$ на $b_0 \ldots b_n$ было упрощено здесь согласно предложению П. Дубоста (P. Dubost) и Ж.-М. Труссе (J.-M. Trousse), доклад STAN-CS-75-511 (Stanford Univ., 1975).]

18. (a) Всего имеется $2^n(m-n)$ символов "$*$", следовательно, $2^n n$ цифр с $2^n n/m$ цифр в каждом столбце. Половина цифр в каждом столбце должна быть равна нулю. Значит, $2^{n-1}n/m$ — целое и в каждом столбце содержится $(2^{n-1}n/m)^2$ несоответствий. Поскольку каждая пара строк имеет минимум одно несоответствие, необходимо получить $2^n(2^n - 1)/2 \leq (2^{n-1}n/m)^2 m$.

(b) Рассмотрим 2^n m-битовых чисел с нулем в $m-n$ определенных столбцах. Половина из них имеет нечетный паритет. Число строк с четным паритетом среди строк с символом "$*$" в любом из неопределенных столбцов равно числу строк с нечетным паритетом.

(c) $*000$, $*111$, $0*10$, $1*10$, $00*1$, $10*1$, $010*$, $110*$. Это построение не такое равномерное, как (13), поскольку запрос типа $*01*$ подходит к четырем строкам, в то время как $*10*$ — только к двум. Заметьте, что (13) имеет циклическую симметрию.

(d) Постройте 4^3 строк из каждой строки (13) путем замещения каждого символа "$*$" символами "$****$", каждого нуля — любой из первых четырех строк и каждой единицы — любой из последних четырех строк (подобное построение создает $\mathrm{ABD}(mm', nn')$ из любых $\mathrm{ABD}(m,n)$ и $\mathrm{ABD}(m',n')$).

(e) Пусть дано $\mathrm{ABD}(16,9)$. Можно в каждой строке заключить в кружок один символ "$*$" таким образом, чтобы в каждом столбце было равное число заключенных в кружки

символов "*". Затем можно разбить каждую строку на две строки, заменяя заключенные в кружки элементы нулями и единицами. Для того чтобы показать, что такая процедура заключения символов "*" в кружки возможна, заметим, что символы "*" в каждом столбце могут быть произвольно разделены на 32 группы по 7 в каждой; тогда 512 строк содержат символы "*" из 7 различных групп каждая и каждая из $32 \times 8 = 512$ групп появляется в 7 различных строках. Теорема 7.5.1E ("теорема о свадьбе") гарантирует существование идеального соответствия одного зацикленного элемента в каждой строке и в каждой группе.

Литература. R. L. Rivest, *SICOMP* **5** (1976), 19–50; A. E. Brouwer, *Combinatorics*, edited by Hajnal and Sós, *Colloq. Math. Soc. János Bolyai* **18** (1978), 173–184. Брувер доказывает, что $\mathrm{ABD}(2n, n)$ существует для всех $n \geq 32$. Метод из п. (d) дает нам также $\mathrm{ABD}(32, 15)$ путем комбинирования (13) и (15).

19. Согласно упр. 8 среднее количество при $8-k$ определенных битах равно $2^{k-3} f_{8-k}(8,8)/\binom{8}{k}$ со значениями соответственно $(32, 22, \frac{104}{7}, \frac{69}{7}, \frac{45}{7}, \frac{33}{8}, \frac{73}{28}, \frac{13}{8}, 1) \approx (32, 22, 14.9, 9.9, 6.4, 4.1, 2.6, 1.6, 1)$ для $8 \geq k \geq 0$. Эти значения лишь несущественно больше значений $32^{k/8} \approx (32, 20.7, 13.5, 8.7, 5.7, 3.7, 2.4, 1.5, 1)$. Значения для наихудшего случая равны $(32, 22, 18, 15, 11, 8, 4, 2, 1)$ (примерами "наихудших запросов" могут служить `********`, `0*******`, `*0****0*`, `**0**0*1`, `**0*00*1`, `*0**0100`, `0001**10`, `000000*0`, `00000000`).

20. В работе J. A. La Poutré, *Disc. Math.* **58** (1986), 205–208, показано, что $\mathrm{ABD}(m, n)$ не может существовать при $m > \binom{n}{2}$ и $n > 3$; следовательно, $\mathrm{ABD}(16, 6)$ не существует. В работе La Poutré and van Lint, *Util. Math.* **31** (1987), 219–225, доказано, что не существует $\mathrm{ABD}(10, 5)$. $\mathrm{ABD}(8, 6)$ получаем из $\mathrm{ABD}(8, 5)$ или $\mathrm{ABD}(4, 3)$, используя метод из упр. 18. Таким образом получается несколько неизоморфных решений, и могут существовать и другие примеры $\mathrm{ABD}(8, 6)$. Единственные остающиеся возможности (кроме тривиальных $\mathrm{ABD}(5, 5)$ и $\mathrm{ABD}(6, 6)$) — $\mathrm{ABD}(8, 5)$, отличное от (15), и, возможно, одно или несколько $\mathrm{ABD}(12, 6)$.

Ладно, я очень рад, что мы до этого сами
додумались, как полагается сыщикам;
за всякий другой способ я гроша ломаного не дам.

— ТОМ СОЙЕР (1884) [Марк Твен, *Приключения Гекльберри Финна*]