С. А. Абрамов

Лекции о сложности алгоритмов

Допущено учебно-методическим советом по прикладной математике и информатике УМО по классическому университетскому образованию в качестве учебного пособия для студентов высших учебных заведений, обучающихся по специальности и направлению «Прикладная математика и информатика» и по направлению «Информационные технологии»

Москва
Издательство МЦНМО
2009

УДК 510.52
ББК 22.12
А16

Научный редактор
Е. А. Бордаченкова

Абрамов С. А.
А16 Лекции о сложности алгоритмов. — М.: МЦНМО, 2009. — 256 с.

ISBN 978-5-94057-433-0

В книге излагаются основные (начальные) разделы теории сложности алгоритмов. Различаются алгебраическая и битовая сложности, каждая из которых рассматривается в худшем случае и в среднем. Ряд основных понятий теории сложности, как-то: оценки снизу и сверху, нижняя граница сложности алгоритмов некоторого класса, оптимальный алгоритм и т. д., рассматривается не только в обычном функциональном, но и в асимптотическом смысле: асимптотические оценки, асимптотическая нижняя граница, оптимальность по порядку сложности и т. д. Показывается, что при исследовании существования алгоритма решения задачи, имеющего «не очень высокую» сложность, важную роль может играть сводимость одной задачи к другой.

Изложение сопровождается анализом сложности большого числа алгоритмов арифметики, сортировки и поиска, вычислительной геометрии, теории графов и др.

Для студентов, специализирующихся в области математики и информатики.

ББК 22.12

ISBN 978-5-94057-433-0

Предисловие

Предлагаемая книга основана на курсе лекций, читаемых автором на факультете вычислительной математики и кибернетики (ВМК) МГУ им. М. В. Ломоносова. Ее цель — обсуждение основных понятий теории сложности и некоторых методов анализа сложности алгоритмов. Это обсуждение сопровождается подробным рассмотрением со сложностной точки зрения ряда алгоритмов арифметики, сортировки и поиска, вычислительной геометрии, теории графов и др. Материал группируется по главам и параграфам в соответствии с разделами самой теории сложности — сложность в худшем случае, сложность в среднем, нижние границы сложности и т. д., а не по тематической принадлежности рассматриваемых алгоритмов — сортировка, теория графов, арифметика и т. д. Для того, чтобы сосредоточиться именно на понятиях и подходах теории сложности, при этом не затрачивая слишком много времени на объяснение деталей трудных для понимания алгоритмов, в примерах исследуются либо алгоритмы достаточно известные (сложностные характеристики которых менее известны), либо алгоритмы, суть которых может быть изложена коротко. Сложностные вопросы рассматриваются в книге довольно подробно, но книга значительно уступает по широте охвата алгоритмического материала книгам Д. Кнута [18, 19, 20], А. Ахо, Дж. Хопкрофта и Дж. Ульмана [5], Т. Кормена, Ч. Лейзерсона, Р. Ривеста [21], С. Баасе и А. ван Гелдера [43], Ж. Брассара и П. Берли [46], в той или иной мере отражающих весь спектр вопросов построения алгоритмов, и книгам по специальным алгоритмическим разделам математики — вычислительной геометрии (Ф. Препарата, М. Шеймос [30]), алгоритмической теории чисел (Э. Бах, Дж. Шаллит [44]), комбинаторики (Э. Рейнгольд, Ю. Нивергельт, Н. Део [31]) и др. Здесь надо сказать, что названная литература, как и некоторые другие издания, послужила источником ряда примеров и задач, предлагаемых в этой книге.

В последних трех параграфах, касающихся классов P и NP, понятия NP-полноты и т. д., ряд фактов дается без доказательства. Это объясняется тем, что в книге используются менее формальные модели вычислений, чем, скажем, машина Тьюринга, а доказательства упомянутых фактов опираются на полностью формализованные мо-

дели. Недостающие доказательства могут быть найдены, например, в [4], [10], [13], [23].

В приложениях даются дополнительные сведения по затрагиваемым в основном тексте вопросам. Исключение составляют приложения A, G: в первом из них содержатся необходимые сведения о ряде алгоритмов сортировки и поиска, во втором — о методе решения линейных рекуррентных уравнений с постоянными коэффициентами; эти два приложения носят характер напоминания.

Библиографические комментарии даются в сносках.

Каждая из глав снабжена задачами для самостоятельного решения, среди которых помимо легких имеются и такие, которые углубляют материал главы, поэтому полезно по крайней мере прочитывать условия всех задач. Задача содержит указание к решению в тех, например, случаях (довольно редких), когда в основном тексте имеется отсылка к этой задаче. По всем главам для задач используется сквозная нумерация. Многие из предлагаемых задач имеют вид утверждений, подразумевается, что каждое такое утверждение надо доказать.

Автор благодарен своим коллегам по ВМК МГУ В. Б. Алексееву и В. П. Иванникову, а также Е. В. Зиме (университет Вилфрид Лауэр, Ватерлоо, Канада) и М. Петковшеку (университет Любляны, Словения), беседы и дискуссии с которыми существенно помогли в отборе материала и выработке общей схемы курса лекций и этой книги, при этом надо особо отметить, что Е. В. Зима принял участие в написании главы 6 и приложения D. Большая благодарность и А. В. Бернштейну (ИСА РАН), А. А. Васину (ВМК МГУ), В. А. Серебрякову (ВЦ РАН), сделавшим полезные замечания по ряду разделов книги. Автор признателен рецензентам М. Н. Вялому (ВЦ РАН) и С. Б. Гашкову (мехмат МГУ) — их пометки на полях рукописи оказали значительную помощь на заключительном этапе работы над книгой. Особая благодарность за советы и многочисленные конструктивные замечания Е. А. Бордаченковой (ВМК МГУ) — научному редактору книги.

Введение

Исследование *сложности* алгоритма помогает понять степень его практической приемлемости. Сравнительный анализ сложности нескольких алгоритмов решения одной и той же задачи позволяет делать обоснованный выбор лучшего из них. Например, если для решения задачи предлагается новый алгоритм, то необходимы доводы, говорящие о его преимуществах в сравнении с известными ранее алгоритмами, и анализ сложности может предоставить такие доводы.

Слово «сложность» в этом контексте является математическим термином, а не общим обозначением препятствия к выполнению замысла. С понятием сложности связываются затраты времени или памяти, соответствующие худшему случаю, либо затраты в среднем; при этом, чтобы обсуждать худший или «средний» случай, нужно, прежде всего, договориться, как определяется *размер входа* (размер входных данных) алгоритма и в чем измеряются *затраты* при работе алгоритма над фиксированным входом.

Часто размер входа определяют как общее число символов в представлении входа, но возможны и другие пути. В задачах сортировки и поиска размер входа — это, как правило, количество элементов n входного массива, в задачах на графах — число вершин $|V|$ или число ребер $|E|$ входного графа $G = (V, E)$, но, с другой стороны, $|V|$ и $|E|$ могут рассматриваться и совместно как два *параметра размера* входа. В арифметических задачах размером входа может быть, например, максимум абсолютных величин входных целых чисел, или количество цифр в двоичной записи этого максимума, или же, как уже говорилось, суммарное количество двоичных цифр всех входных чисел и т. д. — выбор делается в зависимости от характера задачи.

Для алгоритмов сортировки соответствующие затраты времени достаточно полно характеризуются количеством сравнений и перемещений элементов массива. Для алгоритмов на графах учитываемые затраты могут складываться, например, из операций над матрицей смежности или над массивом списков смежных вершин данного графа и из некоторых дополнительных вычислительных операций. Для арифметических алгоритмов в качестве меры затрат может быть взято количество всех выполняемых арифметических операций, или,

как альтернатива, лишь наиболее дорогих операций (например, мультипликативных — умножений и делений); более тщательный анализ требует рассмотрения количества *битовых* операций. Если алгоритм является *рандомизированным* (содержит обращения к генератору случайных чисел с известным распределением), то затраты, вообще говоря, не определяются однозначно входом алгоритма, но зависят от полученных случайных чисел. Можно рассматривать усредненные затраты для каждого конкретного входа; такие затраты уже будут функцией входа.

При фиксированном значении размера входа сами входы алгоритма могут варьироваться, при этом меняются и затраты; алгоритм может быть охарактеризован наибольшими или средними (в соответствии с распределением вероятностей на входах данного размера) затратами. В обоих случаях сложность алгоритма — это функция размера входа или, соответственно, нескольких параметров размера входа. Для этого понятия иногда используется термин *вычислительная сложность*, чтобы избежать путаницы с описательной сложностью, которая тем или иным способом определяется исходя из самой записи (текста) алгоритма; одним из видов описательной сложности является длина записи алгоритма, и именно так понятие сложности трактуется в некоторых теориях[1]. Мы будем рассматривать только вычислительную сложность, называя ее просто сложностью.

Понятие сложности является математическим уточнением довольно расплывчатого термина «трудоемкость», с помощью которого, наряду с не более ясными «быстродействием» и «эффективностью», иногда пытаются характеризовать алгоритмы. Принятие в качестве сложности именно затрат в среднем или затрат, соответствующих худшему (а не, скажем, лучшему) случаю, помогает оценить достаточность имеющихся вычислительных ресурсов для выполнения алгоритма.

Итак, сложность является функцией числового, чаще всего целого, аргумента, иногда — нескольких таких аргументов. Теория сложности изучает эти функции, сопоставленные как отдельным алгоритмам, так и классам алгоритмов решения некоторой задачи. В последнем случае возникает семейство функций, для которого могут обсуждаться вопросы о нахождении *нижней границы*, о существовании минимальной функции этого семейства (если такая функция существует, то соответствующий ей алгоритм — *оптимальный* в рассматриваемом классе) и т. д. Важным является также вопрос о существовании в данном классе алгоритмов, нацеленных на решение конкретной задачи,

[1] См., например, статью «Алгоритма сложность» в [40].

алгоритма такого, сложность которого растет с увеличением размера входа не слишком быстро, например, остается ограниченной некоторым полиномом, вид которого не оговаривается (такой алгоритм принято называть *полиномиальным*).

При исследовании существования алгоритма решения задачи, имеющего «не очень высокую» сложность, важную роль играет *сводимость* какой-то задачи к другой — из возможности быстро решить некоторую задачу может следовать такая же возможность для ряда других, и наоборот, невозможность быстрого решения какой-то одной задачи может автоматически повлечь такую же невозможность для ряда других задач.

Сложности многих алгоритмов трудно или вообще нельзя представить в простом «замкнутом» виде; помимо этого, точное значение сложности алгоритма для каждого конкретного значения размера входа часто не представляет особого интереса, актуальным же является исследование роста сложности при возрастании размера входа (особый интерес к исходным данным большого размера оправдан тем, что на них «захлебываются» тривиальные алгоритмы). Поэтому в теории сложности широко используется *асимптотическое* оценивание. Однако сравнение сложностей различных алгоритмов на основе асимптотических оценок этих сложностей возможно не для всех типов таких оценок.

Этот круг вопросов, наряду с рассмотрением общих достаточно элементарных подходов и методов, которые могут оказаться полезными при сложностном анализе алгоритмов, составляет содержание предлагаемого курса теории сложности алгоритмов.

Наиболее часто используемые обозначения

$C_A^T(x)$, $C_A^S(x)$ — временны́е и пространственные затраты алгоритма A для входа (входных данных) x;

$\|x\|$ — размер входа x;

$T_A(s)$, $S_A(s)$ — значения временно́й и пространственной сложности алгоритма A для значения s размера входа;

$\lambda(n)$ — количество цифр в двоичной записи неотрицательного целого n (битовая длина n);

$\lambda^*(n)$ — количество единиц в двоичной записи неотрицательного целого n;

$\lfloor a \rfloor$ — наибольшее целое, меньшее или равное вещественному a;

$\lceil a \rceil$ — наименьшее целое, большее или равное вещественному a;

$\mathbb{N}$, $\mathbb{N}^+$ — множества неотрицательных и положительных целых чисел;

$\mathbb{Z}$ — множество (кольцо) целых чисел;

$\mathbb{Z}_k$ — кольцо вычетов по модулю k;

$\mathbb{Q}$, $\mathbb{R}$, $\mathbb{C}$ — множества (поля) рациональных, вещественных и комплексных чисел;

Π_n — множество всех перестановок чисел $1, 2, \ldots, n$;

$\square$ — конец доказательства.

Глава 1

Сложности алгоритмов как функции числовых аргументов. Сложность в худшем случае

§ 1. Затраты алгоритма для данного входа, алгебраическая сложность

Пусть по входным данным (входу) x алгоритм A вычисляет результат (выход) y. Такое вычисление связано с затратами времени и памяти. Допустим, что достигнуто соглашение о том, как измеряются эти затраты, тогда можно рассмотреть функции затрат

$$C_A^T(x), \quad C_A^S(x),$$

где верхний индекс T указывает на временные затраты[1], S — на пространственные затраты (затраты памяти и пространственные затраты — это синонимы), соответствующие вычислениям, связанным с применением A к входу x; буквы T и S возникают из английских слов time — время и space — пространство. Вид этих функций может быть очень непростым; их трудно исследовать методами математического анализа, потому что аргумент x не является, вообще говоря, числом и не принадлежит какому-либо метрическому пространству. Можно ввести некоторую неотрицательную числовую характеристику $\|x\|$ возможных входов алгоритма и оценивать функции затрат с помощью функций, зависящих не от x, а от $\|x\|$:

$$C_A^T(x) \leqslant \varphi(\|x\|), \quad C_A^S(x) \leqslant \psi(\|x\|). \tag{1.1}$$

Если φ и ψ не слишком быстро растут при возрастании (числового) аргумента, то эти оценки могут обнадеживать автора или потребителя алгоритма A.

[1] Здесь и далее имеется в виду временны́е (связанные с расходованием времени), а не вре́менные (непостоянные, краткосрочные) затраты. Аналогично — временна́я сложность и т. д.

Избираемую нами величину $\|\cdot\|$ принято называть *размером входа*. В соответствии с нашим замыслом размер входа должен характеризовать «громоздкость» исходных данных; то, что $\|x\|$ является числом, позволяет говорить о росте φ, ψ и исследовать этот рост. Выбор размера входа — существенный этап оценивания функций затрат; нужна, во всяком случае, уверенность, что оценки вида (1.1) будут нести полезную информацию.

В дальнейшем, однако, предметом нашего изучения будут не функции затрат, а некоторые другие функции, о которых мы скажем после ряда примеров. Предварительно условимся о некоторых обычных для литературы по сложности алгоритмов обозначениях. Пусть $a \in \mathbb{R}$, т. е. a — вещественное число. Значение $\lfloor a \rfloor$ есть результат округления a до целого в меньшую сторону: $\lfloor 3{,}14 \rfloor = 3$, $\lfloor -3{,}14 \rfloor = -4$ (эту величину — целую часть a — записывают и как $[a]$). Соответственно, $\lceil a \rceil$ есть результат округления a до целого в большую сторону: $\lceil 3{,}14 \rceil = 4$, $\lceil -3{,}14 \rceil = -3$. Если a — целое, то $\lfloor a \rfloor = \lceil a \rceil = a$.

Рассмотрим три простых примера.

Пример 1.1. Исходя из определения произведения матриц, мы получаем алгоритм умножения двух квадратных матриц порядка n, требующий n^3 операций умножения. Этот алгоритм далее будет обозначаться буквами MM, от английских слов matrix multiplication — матричное умножение.

Пример 1.2. Один из очевидных алгоритмов распознавания простоты данного $n \in \mathbb{N}^+$, который мы будем называть алгоритмом пробных делений, состоит в выяснении того, имеется ли среди чисел $2, 3, \ldots, \lfloor \sqrt{n} \rfloor$ хотя бы одно, делящее n. Количество проверок делимости n на те или иные целые числа (для краткости можно говорить о «числе делений») колеблется от 1 до $\lfloor \sqrt{n} \rfloor - 1$. Этот алгоритм далее будет обозначаться буквами TD, от английских слов trial divisions — пробные деления.

Пример 1.3. При сортировке простыми вставками число сравнений[1] колеблется от $n-1$ до $\frac{n(n-1)}{2}$, в зависимости от входного массива $x_1, x_2, \ldots, x_n$. Если интересоваться числом перемещений элемен-

[1] Основные алгоритмы сортировки и поиска приводятся в приложении А. Обсуждая конкретные алгоритмы сортировки, мы для краткости иногда будем опускать само слово «алгоритм», говоря, например, не об алгоритме сортировки простыми вставками, а просто о сортировке простыми вставками и т. д. Все сортировки, которые мы рассматриваем, являются сортировками с помощью сравнений, т. е. никаких других операций, кроме сравнений и перемещений (присваиваний или обменов), над элементами массива производиться не может, — исключение составляет пример 29.1. Элементы сор-

тов (операций обмена $\leftrightarrow$), то оно колеблется от 0 до $\frac{n(n-1)}{2}$. Эта сортировка далее будет обозначаться буквой I, от английского слова insert — вставка.

В этих примерах уже фактически сказано, что считается размером входа и затратами. В примере 1.1 размер входа — целое неотрицательное n — это порядок матриц. В примере 1.2 размер входа — само входное число (сам вход) n. В примерах 1.1, 1.2 количество исследуемых операций однозначно определяется по n. Столь жесткая зависимость затрат от размера входа может и не иметь места, как это следует из примера 1.3, где размер входа — это длина n массива. В связи с этим примером остановимся пока на операциях сравнения. Количество сравнений $\frac{n(n-1)}{2}$, требуемое в худшем случае, характеризует рассматриваемый алгоритм среди всех алгоритмов сортировки как «имеющий квадратичную сложность». Для придания этим словам точного смысла должно быть, прежде всего, определено само понятие сложности.

Определение 1.1. Пусть на возможных входах x алгоритма A определена неотрицательная числовая функция $\|x\|$ (размер входа). Пусть также определены целочисленные неотрицательные функции $C_A^T(x)$, $C_A^S(x)$ временных и пространственных затрат алгоритма A. Тогда *временной и пространственной сложностями* A называются функции числового аргумента

$$T_A(n) = \max_{\|x\|=n} C_A^T(x), \quad S_A(n) = \max_{\|x\|=n} C_A^S(x) \tag{1.2}$$

(областью изменения n как аргумента функций $T_A(n)$ и $S_A(n)$ является множество значений размера $\|\cdot\|$). Более полно каждая такая сложность именуется сложностью *в худшем случае*.

Вместо $T_A(n), S_A(n)$ мы иногда будем писать просто $T(n), S(n)$, если ясно, о каком алгоритме идет речь.

Два примечания к определению 1.1.

1) Значение $C_A^T(x)$ есть в подавляющем большинстве случаев число каких-то операций, а $C_A^S(x)$ — число хранимых величин какого-то типа, поэтому предположение о целочисленности значений этих функций вполне естественно. Без этого предположения в (1.2) пришлось

тируемых массивов считаются попарно различными. Если не оговорено противное, то речь всегда идет о сортировке по возрастанию. Размером входа каждой из сортировок мы без специального упоминания считаем число n элементов массива.

бы использовать sup вместо max, что сделало бы многие рассуждения о сложности более тяжеловесными.

2) Видимо, вместо «сложность в худшем случае» правильнее было бы сказать «сложность как затраты в худшем случае», но такова принятая здесь терминология. Мы пока (до § 5) не рассматриваем сложность в среднем, и поэтому, не опасаясь путаницы, будем называть сложность в худшем случае просто сложностью.

Вернемся к алгоритмам MM, TD, I из примеров 1.1, 1.2, 1.3. Рассмотрим вначале временные сложности $T_{\mathrm{MM}}(n), T_{\mathrm{TD}}(n), T_{\mathrm{I}}(n)$. Получаем $T_{\mathrm{MM}}(n)=n^3$ и $T_{\mathrm{I}}(n)=\dfrac{n(n-1)}{2}$. Для $T_{\mathrm{TD}}(n)$ у нас нет столь простой формулы. Заметим, например, что для значений n от 111 до 120 мы имеем

n:	111	112	113	114	115	116	117	118	119	120
$T_{\mathrm{TD}}(n)$:	2	1	9	1	4	1	2	1	6	1

Изменение этой функции при $n\to\infty$ можно охарактеризовать так: для любых n значение этой функции не превосходит $\lfloor\sqrt{n}\rfloor-1$, и найдутся сколь угодно большие n, для которых ее значение равно $\lfloor\sqrt{n}\rfloor-1$.

В примерах 1.2 и 1.3 не возникает необходимости в хранении больших объемов величин сверх объема входа (сортировка простыми вставками не требует дополнительного массива). Дополнительная память, если и нужна, — это несколько дополнительных переменных (ячеек), используемых алгоритмом.

Поэтому можно считать, что $S_{\mathrm{I}}(n), S_{\mathrm{TD}}(n)$ ограничены константами. В примере 1.1 нам требуется дополнительная матрица, в которой будет формироваться результат; имеем $S_{\mathrm{MM}}(n)=n^2$.

Во всех этих примерах мы, как легко заметить, принимаем во внимание лишь часть затрат и в соответствии с этим определяем сложность алгоритма. В примере 1.1 учитывались только умножения как более дорогие (доминирующие по затратности) операции. В примере 1.3 мы рассматриваем либо сравнения, либо обмены. И во всех примерах мы игнорировали затраты по выполнению управляющих операторов, хотя, скажем, при выполнении простейшего оператора цикла

```
for i = 1 to n do ... od
```

тратится время на изменение параметра цикла, проверку условия продолжения цикла и т. д.

Обычно в таких ситуациях считается, что мы учитываем основную часть затрат для худшего случая. Если пытаться точно подсчитывать

все, не упуская ничего, то получение и обоснование оценки сложности заурядного алгоритма может превратиться в очень тяжелую работу. Принимая решение о том, в чем измеряются временные затраты, мы разделяем важное и неважное. Разумеется, при этом необходима осторожность, без которой к разряду неучитываемых «дешевых» операций могут быть отнесены такие, на выполнение которых в реальных вычислениях уйдет существенное время из-за их огромного количества.

Отметим еще, что, вообще говоря, время выполнения операции зависит от размеров ее операндов, это относится и к арифметическим операциям, и к операции сравнения (например, длинные числа сравнивать труднее, чем короткие и т. д.). Аналогично дело обстоит с пространственными затратами и соответственно с пространственной сложностью; здесь, прежде чем находить сложность, надо решить, что является «единицей хранения»: скажем, при умножении двух матриц мы можем получить матрицу с элементами, которые записываются более длинно, чем элементы исходных матриц. Однако довольно часто считается, что результат операции всегда умещается в одной ячейке.

Иногда приходится отвлекаться от того, что десятичная или двоичная запись иррационального числа не может быть размещена в ячейке какого-либо устройства. Более того, в теории алгебраической сложности как разделе алгебры [1] рассматривают, например, алгоритмы, применимые к элементам произвольного абстрактного кольца или поля, и вопрос о представлении этих элементов в реальном компьютере или в абстрактной машине не обсуждается вообще.

Определение 1.2. Пусть функция $C_A^T(x)$ в определении 1.1 отражает затраты на выполнение лишь какого-то одного типа операций в предположении, что все эти операции требуют одинакового времени выполнения, не зависящего от вида и размера операндов, и это время равно 1. Тогда соответствующая функция $T_A(n)$ — это сложность *по числу операций* рассматриваемого типа. Привлечение в качестве $C_A^S(x)$ функции, отражающей только количество хранимых дополнительных величин того или иного фиксированного типа, приводит к пространственной сложности $S_A(n)$ *по числу величин* рассматриваемого типа.

Такую сложность называют *алгебраической* сложностью по числу операций или числу величин рассматриваемого типа, подчерки-

[1] См., например, [25].

вая этим предположение о равнозатратности всех рассматриваемых операций и равнообъемности всех учитываемых величин. Алгебраическую сложность арифметических алгоритмов по числу умножений и делений называют *мультипликативной* сложностью. Из основных арифметических операций умножение и деление являются наиболее дорогими, и при рассмотрении арифметических алгоритмов часто интересуются именно (алгебраической) мультипликативной сложностью.

Несколько позднее мы будем говорить о *битовой* сложности, которая позволяет принимать в расчет различие в размерах операндов и рассматривать при нахождении сложности непохожие операции, сопоставляя их затратность на битовом уровне.

Формулы (1.2) вновь заставляют вспомнить о важности понятия размера входа. В силу разных причин не всегда этот размер легко и естественно может быть введен так, что сложность алгоритма возрастает при увеличении размера входа. Но, по крайней мере, функция $\|\cdot\|$ должна обеспечивать определенность значений $T_A(n)$, $S_A(n)$ для всех достаточно больших n — иначе было бы невозможно говорить о поведении $T_A(n)$, $S_A(n)$ при $n \to \infty$.

Заведомо неудачный размер входа — это, скажем, число строк n данной матрицы, если речь идет об алгоритмах вычисления произведения всех элементов и если матрица не обязательно является квадратной: при выбранном таким образом размере входа мы имеем тождественное равенство $T_A(n) = \infty$ для сложности по числу умножений любого такого алгоритма A, так как при фиксированном числе строк число столбцов и число элементов могут быть сколь угодно большими.

Другие возможные осложнения, вызванные неудачным выбором размера входа, будут обсуждаться в § 13.

Достаточно очевидно, что если алгоритм A состоит в последовательном (друг за другом) выполнении алгоритмов U и V, то мы, вообще говоря, не можем утверждать, что $T_A(n) = T_U(n) + T_V(n)$, так как, например, размер входа алгоритма U может существенно отличаться, как в большую, так и в меньшую сторону, от размера его выхода.

Пусть A и B — некоторые алгоритмы. Как можно заметить, из выполнения для всех n неравенства $T_A(n) < T_B(n)$ не следует, что $C_A^T(x) < C_B^T(x)$ для всех возможных входов x (достаточно вспомнить сортировку слияниями и пузырьковую сортировку; последняя выполняется быстрее первой, коль скоро массив задан уже в упорядоченном виде). Аналогично дело обстоит с пространственными сложностью и затратами.

В некоторых случаях алгоритму сопоставляют несколько сложностей. Итоговые временные затраты сортировки массива с помощью сравнений складываются из затрат на сравнение и на перемещение элементов. Без учета типа элементов массива нельзя сказать, какая из операций является более дорогой. Поэтому для алгоритмов сортировки используются две сложности: с одной стороны — по числу сравнений, а с другой — по числу перемещений элементов, т. е. присваиваний (:=) или обменов ($\leftrightarrow$). Если же рассматривается некоторый арифметический алгоритм, то, исследовав его мультипликативную сложность, можно дополнительно интересоваться, в качестве уточнения, *аддитивной* сложностью (числом сложений и вычитаний в худшем случае) и т. д. В приложении D мы тщательно исследуем мультипликативную и аддитивную сложности алгоритмов вычисления значения полинома при заданном значении аргумента.

Пусть некоторому алгоритму A сопоставлены две, скажем, временные сложности $T'_A(n)$ и $T''_A(n)$. Например, $T'_A(n)$ и $T''_A(n)$ могут быть сложностями по разным операциям. Если операции первого и второго вида предполагаются имеющими одинаковую затратность, то это еще не дает оснований считать, что сложность $\widetilde{T}_A(n)$, которую мы определим, рассматривая совместно операции обоих видов, будет равна $T'_A(n) + T''_A(n)$, потому что максимумы этих двух видов затрат могут достигаться на разных входах одного и того же размера. Можно только утверждать, что

$$\widetilde{T}_A(n) \leqslant T'_A(n) + T''_A(n).$$

Пример 1.4. Чтобы проиллюстрировать указанное обстоятельство, мы вновь обратимся к сортировке простыми вставками, которая может рассматриваться в двух вариантах I_1 и I_2: для вставки элемента x_{i+1} в уже упорядоченную часть $x_1, x_2, \ldots, x_i$ элементы этой упорядоченной части просматриваются либо в порядке $x_i, x_{i-1}, \ldots$, либо в порядке $x_1, x_2, \ldots$ В первом варианте максимум числа сравнений достигается тогда, когда входной массив имеет обратную к требуемой упорядоченность: $x_1 > x_2 > \ldots > x_n$, и на этом же входе достигается максимум числа обменов; имеем $\widetilde{T}_{\mathrm{I}_1}(n) = T'_{\mathrm{I}_1}(n) + T''_{\mathrm{I}_1}(n) = n(n-1)$, где $T'_{\mathrm{I}_1}(n), T''_{\mathrm{I}_1}(n)$ суть сложности по числу сравнений и обменов.

Во втором варианте максимум числа сравнений достигается, когда массив уже упорядочен так, как требуется: $x_1 < x_2 < \ldots < x_n$, а максимум числа обменов — когда $x_1 > x_2 > \ldots > x_n$. Если $i > 1$, то при вставке элемента x_{i+1} в уже упорядоченную часть $x_1, x_2, \ldots, x_i$ потребуется тем меньше обменов, чем больше потребуется сравнений. Если элемент

займет место с номером k, то это потребует $i-k+1$ обменов. Число же сравнений равно k, если $k \leqslant i$, и равно $k-1$, если $k = i+1$. Суммарное число сравнений и обменов равно $i+1$, если $k \leqslant i$, и равно i, если $k = i+1$. Поэтому максимум общего числа сравнений и обменов достигается, например, на входном массиве, имеющем обратную к требуемой упорядоченность: $x_1 > x_2 > \ldots > x_n$. Легко устанавливается, что $\widetilde{T}_{\mathrm{I}_2}(n) = \dfrac{(n+2)(n-1)}{2}$.

Рассматривая сложности $\widetilde{T}_{\mathrm{I}_1}(n)$, $\widetilde{T}_{\mathrm{I}_2}(n)$ первого и второго вариантов сортировки простыми вставками по общему числу сравнений и обменов, мы имеем

$$\widetilde{T}_{\mathrm{I}_1}(n) = n(n-1), \quad \widetilde{T}_{\mathrm{I}_2} = \frac{(n+2)(n-1)}{2}, \tag{1.3}$$

и, таким образом, $\widetilde{T}_{\mathrm{I}_1}(n)/\widetilde{T}_{\mathrm{I}_2}(n) \to 2$ при возрастании n.

Обладающий большей общей сложностью $\widetilde{T}_{\mathrm{I}_1}$ первый вариант алгоритма рассматривается в литературе значительно чаще второго, не в последнюю очередь из-за того, что его запись несколько проще: мы можем со всеми подробностями записать первый вариант с помощью псевдокода

```
for i = 2 to n do
    k := i;
    while k > 0 and x_k > x_{k+1} do x_k ↔ x_{k+1}; k := k - 1 od
od
```

(предполагается, что если $k \leqslant 0$, то булево выражение, имеющее вид конъюнкции

$$k > 0 \text{ and } x_k > x_{k+1},$$

принимает значение 0, т. е. «ложь», даже если при этом, например, значение x_k не определено, и, следовательно, не определено значение второго члена конъюнкции).

Второй вариант будет иметь вид:

```
for i = 2 to n do
    k := 1;
    while k < i and x_k < x_i do k := k + 1 od;
    for j = i - 1 downto k do x_j ↔ x_{j+1} od
od
```

В первом варианте используется на один оператор цикла меньше, но с точки зрения вычислительной сложности это не является пре-

имуществом, — важным является число лишь учитываемых операций при выполнении алгоритма. Хотя обычно для каждой сортировки рассматривают сложности по сравнениям и перемещениям раздельно, но, тем не менее, если эти сложности совпадают с соответствующими сложностями другой сортировки, то совместное рассмотрение обоих типов операций может оказаться небезынтересным.

Заметим попутно, что среди сортировок с квадратичной сложностью по числу сравнений чрезвычайно низкой сложностью по числу перемещений, а именно сложностью, равной $n - 1$, обладает сортировка выбором.

В некоторых случаях к учитываемым операциям относят операции весьма разнородные, и время выполнения каждой из них считают равным 1 — пример 3.1, как и некоторые другие, даст соответствующую иллюстрацию. При нахождении каждой такой «смешанной» сложности затраты для каждого конкретного входа учитываются разом все вместе.

Этот параграф завершим обсуждением тех предположений о выполнении рассматриваемых алгоритмов, которые неявно уже использовались нами. Мы исходили из того, что вычисления проводятся некоторой машиной, память которой состоит из ячеек. При этом как объем каждой ячейки, так и общее число ячеек считается бесконечным, что, разумеется, является чистой абстракцией. Поместив в ячейку результат каких-то вычислений или считав содержимое некоторой ячейки, машина обращается еще к какой-то ячейке и т. д. Относительно этого процесса предполагается, что сам переход от ячейки к ячейке не связан с какими-либо затратами. Этот принцип лежит в основе модели вычислений, называемой RAM («равнодоступная адресная машина» — довольно неуклюжий, но принятый термин; имеется в виду машина с равнодоступными ячейками памяти) [1]. Модель RAM существенно отличается в этом смысле от другой известной модели вычислений — машины Тьюринга (МТ), где время перехода от одной ячейки ленты к другой зависит от расстояния между ячейками; при этом одна ячейка ленты МТ содержит один символ фиксированного конечного алфавита [2].

[1] В литературе на английском языке — RAM (random-access machine).

[2] Раньше в литературе по вычислительной математике и программированию слово «машина» употреблялось как обозначение некоторого реального вычислительного устройства; устойчивым словосочетанием было, например, «электронная вычислительная машина» (ЭВМ). Сейчас в этом значении в основном употребляется слово «компьютер», а слово «машина» часто используется при обсуждении абстрактных вычислительных устройств: машин Тьюринга, равнодоступных адресных машин и т. д.

Употребление слова «модель» в этом контексте подчеркивает то обстоятельство, что реальные вычислительные устройства (компьютеры) работают на тех или иных специфических принципах, но при исследовании вычислительной сложности алгоритмов этой спецификой можно до известного предела пренебрегать и исходить из упрощающей системы допущений.

Далее мы будем еще неоднократно обращаться к понятию модели вычислений; всюду до главы 5 без оговорок подразумевается модель PAM.

§ 2. Асимптотические оценки (формализм)

Для характеристики роста сложности по числу сравнений сортировки простыми вставками первого и второго типа мы можем использовать известный из математического анализа символ O и написать $T(n)=O(n^2)$ (здесь и далее $n\to\infty$). Мы можем также выделить главные по росту слагаемые в (1.3):

$$\widetilde{T}_{\mathrm{I}_1}(n)=n^2+O(n),\quad \widetilde{T}_{\mathrm{I}_2}(n)=\frac{1}{2}n^2+O(n), \tag{2.1}$$

хотя в этом и нет ощутимого практического смысла в силу простоты функций $\widetilde{T}_{\mathrm{I}_1}(n), \widetilde{T}_{\mathrm{I}_2}(n)$. В то же время, как это хорошо известно, асимптотическая формула $f(n)=O(g(n))$ является удобным средством оценивания нетривиально устроенной функции $f(n)$ с помощью более простой функции $g(n)$; столь же полезными оказываются и оценки вида $f(n)=o(g(n))$. Но когда мы говорим, что сортировка выбором, пузырьковая сортировка и сортировка простыми вставками имеют квадратичные сложности, мы имеем в виду не только то, что соответствующие сложности допускают оценку $O(n^2)$, но что эти сложности являются *величинами порядка* n^2; в математическом анализе это иногда записывается как $T(n)\asymp n^2$, где $T(n)$ — рассматриваемая функция, в данном случае — сложность. В последние годы в теории сложности алгоритмов вместо $f(n)\asymp g(n)$ стали писать $f(n)=\Theta(g(n))$.

Определение 2.1. Функции $f(n)$ и $g(n)$ имеют *одинаковый порядок* (пишут $f(n)=\Theta(g(n))$) тогда и только тогда, когда найдутся положительные c_1, c_2, N такие, что неравенства

$$c_1|g(n)|\leqslant|f(n)|\leqslant c_2|g(n)| \tag{2.2}$$

выполнены для всех $n>N$.

Без труда проверяется, что отношение «иметь одинаковый порядок» является отношением эквивалентности на множестве функций,

определенных для всех достаточно больших значений n (в нашем случае эти значения целые). Несимметричность записи $f(n) = \Theta(g(n))$ в сравнении с записью $f(n) \asymp g(n)$ ($f(n)$ и $g(n)$ как бы не равноправны в первой записи, хотя имеем дело с отношением эквивалентности) объясняется тем, что обычно эту запись используют, когда $g(n)$ проще, чем $f(n)$.

Итак, для сложности $T(n)$ по числу сравнений для любого из упомянутых алгоритмов сортировки мы имеем $T(n) = \Theta(n^2)$. Это более сильное утверждение, чем $T(n) = O(n^2)$, так как $T(n) = O(n^2)$ является лишь асимптотической верхней оценкой: в соответствии с известным из математического анализа определением

$$f(n) = O(g(n)) \iff \exists_{c,N>0}\ \forall_{n>N}\ |f(n)| \leqslant c|g(n)|,$$

например, $n = O(n^2)$, но неверно, что $n = \Theta(n^2)$. Здесь и далее, пользуясь кванторами $\exists$, $\forall$, мы записываем связываемые ими переменные, равно как и условия, определяющие множества значений этих переменных, в виде индексных выражений при кванторах. Это часто позволяет обходиться без дополнительных скобок и облегчает чтение формул.

Иногда бывают полезными нижние асимптотические оценки.

Определение 2.2. Соотношение $f(n) = \Omega(g(n))$ имеет место тогда и только тогда, когда найдутся положительные c, N такие, что для всех $n > N$ выполнено $|f(n)| \geqslant c|g(n)|$.

Следующее предложение выводится из определений символов O, Ω и Θ.

Предложение 2.1. *Соотношение $f(n) = \Theta(g(n))$ имеет место тогда и только тогда, когда одновременно $f(n) = O(g(n))$ и $f(n) = \Omega(g(n))$; помимо этого, $f(n) = \Omega(g(n))$ тогда и только тогда, когда $g(n) = O(f(n))$.*

Если размер входа является целым положительным числом, то возникающие функции являются последовательностями. Для единообразия мы, как правило, будем говорить о функциях, подразумевая, но не упоминая специально, что каждая такая функция определена лишь для целых положительных значений аргумента (возможно даже, только для достаточно больших целых положительных значений аргумента). Итак, при $n \to \infty$ оценки вида $f(n) = \Lambda(g(n))$, где Λ — один из символов Ω, O, Θ, предполагают, что функции $f(n), g(n)$ определены для всех достаточно больших n. Соответствующее неравенство из

числа

$$|f(n)| \geqslant c_1|g(n)|, \quad |f(n)| \leqslant c_2|g(n)|, \quad c_1|g(n)| \leqslant |f(n)| \leqslant c_2|g(n)| \qquad (2.3)$$

тоже, в соответствии с определением, должно выполняться лишь для n, больших некоторого N. Заметим, однако, что если $f(n)$ и $g(n)$ определены для всех $n \in \mathbb{N}^+$ и принимают при $1 \leqslant n \leqslant N$ ненулевые значения, то можно считать, что соответствующее неравенство из перечисленных в (2.3) выполняется для всех n, так как, положив

$$m = \min_{1 \leqslant n \leqslant N} \frac{|f(n)|}{|g(n)|}, \quad M = \max_{1 \leqslant n \leqslant N} \frac{|f(n)|}{|g(n)|},$$

мы можем заменить c_1, c_2 в (2.3) на $c_1' = \min\{c_1, m\}$, $c_2' = \min\{c_2, M\}$. Это замечание в некоторых случаях будет для нас полезным.

Вернемся к примеру 1.2. Для сложности алгоритма пробных делений было бы ошибкой утверждать, что его сложность по числу делений есть $\Theta(\sqrt{n})$. Но оценка $O(\sqrt{n})$, разумеется, верна и, более того, является точной в смысле следующего определения.

Определение 2.3. Если имеет место оценка $f(n) = O(g(n))$, то она называется *точной*, коль скоро существует неограниченно возрастающая последовательность неотрицательных целых чисел $\{n_k\}$ такая, что для $\varphi(k) = f(n_k)$, $\psi(k) = g(n_k)$ имеет место $\varphi(k) = \Omega(\psi(k))$.

Для упомянутых $\varphi(k)$ и $\psi(k)$ в силу этого определения и семантики символа Θ выполнено $\varphi(k) = \Theta(\psi(k))$.

При рассмотрении алгоритма пробных делений для доказательства точности оценки $O(\sqrt{n})$ можно взять n_k равным k-му простому числу, $k = 1, 2, \ldots$

Понятие точности оценки вида $f(n) = O(g(n))$ можно определить также с помощью знакомого из математического анализа символа o; напомним, что $u(n) = o(v(n))$ при $n \to \infty$, коль скоро $u(n) = \alpha(n)v(n)$ и $\lim\limits_{n \to \infty} \alpha(n) = 0$.

Предложение 2.2. *Пусть $f(n) = O(g(n))$. Эта оценка является точной, если и только если неверно, что $f(n) = o(g(n))$.*

Доказательство. Пусть оценка является точной, и $\{n_k\}$ — возрастающая последовательность, о которой говорится в определении 2.3. Тогда существует положительная константа c такая, что $|f(n_k)| \geqslant c|g(n_k)|$, $k = 1, 2, \ldots$, и соотношение $f(n) = o(g(n))$ места не имеет. Обратно, если неверно, что $f(n) = o(g(n))$, то по определению символа o существуют $\varepsilon > 0$ и возрастающая последовательность $\{n_k\}$ натуральных чисел такие, что $|f(n_k)| \geqslant \varepsilon|g(n_k)|$, $k = 1, 2, \ldots$ Если при

этом выполнена оценка $f(n) = O(g(n))$, то эта оценка точна в соответствии с определением 2.3. □

Для рассматриваемой сложности алгоритма пробных делений не верна, скажем, оценка $O(\log n)$, потому что для этой сложности оценка $O(\sqrt{n})$ является точной и в то же время $\log n = o(\sqrt{n})$.

Нелишним будет заметить, что сложность алгоритма пробных делений допускает оценки $O(n)$, $O(n^5)$, $O(n \log n)$ и т. д., хотя, разумеется, эти оценки являются более грубыми в сравнении с $O(\sqrt{n})$. Еще раз подчеркнем, что оценка $f(n) = O(g(n))$ есть асимптотическая *верхняя* оценка, равно как оценка $f(n) = \Omega(g(n))$ — асимптотическая *нижняя* [1]. Как, например, из $l < 5$ и $m < 100$ нельзя вывести, что $l < m$, так и из $f(n) = O(n^2)$, $g(n) = O(n^3)$ нельзя вывести, что хотя бы для достаточно больших n выполняется $f(n) < g(n)$. Оценка вида $T_A(n) = O(g(n))$ (или $S_A(n) = O(g(n))$) подходит для того, чтобы «похвалить» алгоритм A, т. е. охарактеризовать его сложность как достаточно низкую (речь идет лишь об оценках вида $O(g(n))$, а не о более тонких оценках, включающих символ O и имеющих вид, подобный (2.1)), но не для того, чтобы «раскритиковать» его — для таких целей скорее подойдет оценка вида $T_A(n) = \Omega(h(n))$. Зная, например, что сложность по числу обменов для сортировки выбором есть $O(n)$, а для сортировки простыми вставками — $\Omega(n^2)$, мы обоснованно заключаем, что для достаточно больших n первая сложность меньше второй.

Оценки вида $T_A(n) = \Theta(g(n))$, соединяющие в себе оценки $T_A(n) = O(g(n))$ и $T_A(n) = \Omega(g(n))$, в соответствующих ситуациях подходят и для характеризации сложности как сравнительно низкой, и, наоборот, как сравнительно высокой [2].

При всем этом, иногда можно услышать сообщения о новых алгоритмах, сопровождаемые рассуждениями в духе следующего (подразумевается, что n — размер входа): «Лучший из известных ранее алгоритмов решения этой задачи требует $O(n^3)$ операций, а пред-

[1] В книге [6] отмечено, что положение с символом O схоже с тем, которое возникнет, если кто-нибудь «вместо слов „меньше чем“ начнет писать $= M$, например, так: $3 = M(5)$. На вопрос: „Что значит $M(5)$?“ — он должен ответить: „Нечто, что меньше, чем 5“. Таким образом, он быстро привыкает читать M как „нечто, что меньше, чем“, приближаясь к тем самым словам, которые употребляем мы, вводя соотношение $f(s) = O(\varphi(s))$».

[2] Θ- и Ω-нотации вошли в литературу по вычислительной сложности алгоритмов с появлением статьи Д. Кнута [52], в которой автор, в частности, пишет о бессмысленности нижних оценок вида $O(f(n))$ и о невозможности использования оценок такого рода как оценок сложности при сравнении алгоритмов. В [52] отмечается также, что Ω-нотация использовалась ранее в работах Э. Ч. Титчмарша, известного математика первой половины XX века.

лагаемый нами алгоритм — лишь $O(n)$. Таким образом, достигнуто улучшение на два порядка по числу операций». Но информация, содержащаяся в первой из этих двух фраз, не дает достаточных оснований для сделанного заключения. Более того, на основе этой информации вообще нельзя сказать, какой из двух алгоритмов — известный ранее или новый — требует меньше операций при больших n, ведь речь идет лишь об оценках сверху, и возможно, что первая из них может быть улучшена.

Если про оценку $O(g(n))$ известно, что она точная, то это расширяет возможности ее использования. Допустим, что нам известен алгоритм распознавания простоты числа n, имеющий мультипликативную сложность $O(\log^d n)$ при некотором $d > 0$. Тогда мультипликативная сложность этого алгоритма для бесконечного множества значений n (но, может быть, не для всех n) будет меньше, чем мультипликативная сложность алгоритма пробных делений, и для этого вывода достаточно того, что сложность алгоритма пробных делений допускает точную оценку $O(\sqrt{n})$.

В тех случаях, когда рассматриваются два или более параметров размера входа, мы можем по-прежнему использовать асимптотические оценки вида $\Theta(g(n_1, n_2))$, где под знаком Θ расположена функция двух переменных n_1, n_2, причем $n_1, n_2 \to \infty$; определение Θ легко модифицируется на случай двух и большего числа переменных:

$$f(n_1, n_2) = \Theta(g(n_1, n_2)) \iff$$
$$\iff \exists_{c_1, c_2, N>0}\ \forall_{n_1, n_2 > N}\ c_1 |g(n_1, n_2)| \leqslant |f(n_1, n_2)| \leqslant c_2 |g(n_1, n_2)|. \quad (2.4)$$

То же самое с Ω, O и o. При этом, если имеет место оценка $f(n_1, n_2) = = O(g(n_1, n_2))$, то мы назовем ее точной, коль скоро неверно, что $f(n_1, n_2) = o(g(n_1, n_2))$.

Утверждение, что $f(n)$ и $g(n)$ асимптотически эквивалентны, записываемое как $f(n) \sim g(n)$, означает, как известно, что $f(n) = g(n) + + o(g(n)) = g(n)(1 + o(1))$. Утверждение, что $f(n) \sim g(n)$, является, очевидно, более сильным, чем утверждение, что $f(n) = \Theta(g(n))$. Заметим кстати, что из формул (2.1) следует

$$\widetilde{T}_{\mathrm{I}_1}(n) \sim n^2, \quad \widetilde{T}_{\mathrm{I}_2}(n) \sim \frac{1}{2} n^2 \quad (2.5)$$

(например, $\widetilde{T}_{\mathrm{I}_1}(n) = n^2 + O(n) = n^2\left(1 + O\left(\frac{1}{n}\right)\right) = n^2(1 + o(1))$), но не наоборот. Из (2.5) следует только, что

$$\widetilde{T}_{\mathrm{I}_1}(n) = n^2 + o(n^2), \quad \widetilde{T}_{\mathrm{I}_2}(n) = \frac{1}{2} n^2 + o(n^2),$$

при этом из $v(n) = o(n^2)$ не следует, что $v(n) = O(n)$, что доказывается примером $v(n) = n^{3/2}$.

Слова «$f(n)$ имеет асимптотику $g(n)$» означают, что $f(n) \sim g(n)$; например, $\widetilde{T}_{\mathrm{I}_1}(n)$ имеет асимптотику n^2, а $\widetilde{T}_{\mathrm{I}_2}(n)$ имеет асимптотику $\frac{1}{2}n^2$.

Сложности многих алгоритмов трудно или невозможно представить элементарного вида функциями от размера входа. Помимо этого, точное значение сложности алгоритма для каждого конкретного значения размера входа часто не представляет особого интереса, актуальным же является исследование роста сложности при возрастании размера входа. Поэтому асимптотическое оценивание широко используется в теории сложности.

§ 3. Асимптотические оценки (два примера)

Если мы изначально имеем эскизное описание алгоритма, не содержащее мелких деталей, но полностью отражающее его идею, то уже этого эскиза может быть достаточно для получения некоторой информативной асимптотической оценки сложности; проработка деталей алгоритма будет влиять на скрытые за символами O, Ω, Θ значения констант.

Пример 3.1. Займемся задачей построения *выпуклой оболочки* конечного множества M точек координатной плоскости, т. е. выпуклого многоугольника H, содержащего все множество M (рис. 1). Множест-

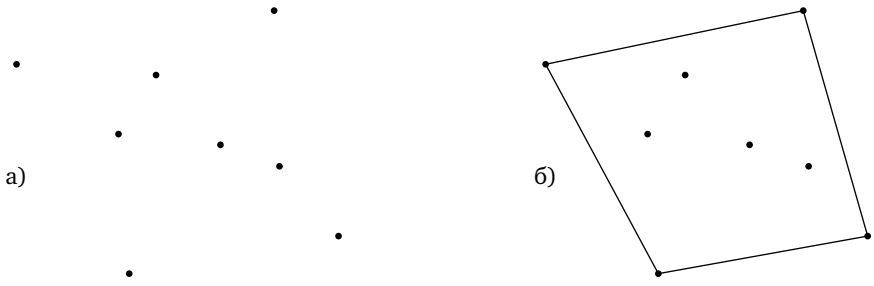


Рис. 1. а) Конечное множество M точек плоскости; б) выпуклая оболочка множества M.

во M задается массивом координат принадлежащих ему точек; требуется построить массив координат вершин многоугольника H при обходе этого многоугольника, начиная с какой-нибудь его вершины,

против часовой стрелки[1] (считаем, что это направление совпадает с направлением обхода точек $(0,0)$, $(1,0)$, $(0,1)$, $(0,0)$).

Пусть n — число элементов множества M, будем считать это число размером входа. Алгоритм, основанный на переборе всех подмножеств множества M с проверкой для каждого из них, является ли оно множеством вершин искомого многоугольника H, имеет очень высокую сложность $\Omega(2^n)$. Обсудим идею значительно более экономного алгоритма Р. Л. Грэхема (этот алгоритм мы обозначим буквой G).

Можно довольно быстро найти среди точек множества M такую, которая обязательно будет одной из вершин многоугольника H: достаточно выбрать в M точку P с наименьшей ординатой, а если таких точек несколько, то из этих нескольких взять ту, которая имеет наименьшую абсциссу. Дополнительно найдем точку O, которая принадлежит многоугольнику H, но не совпадает ни с одной из точек множества M: возьмем для этого какие-нибудь две точки из M и найдем середину соединяющего их отрезка (если впоследствии вдруг окажется, что эта точка принадлежит M, то можно будет удалить ее из M, так как она заведомо не является вершиной H).

Используя какую-нибудь сортировку с помощью сравнений, все точки множества M можно упорядочить по возрастанию углов между отрезком OP и отрезками, соединяющими O с точками множества M, при этом мы считаем, что величина каждого угла принадлежит полуинтервалу $[0, 2\pi)$. Если вдруг обнаружится, что два каких-то угла равны, то упорядочим соответствующие точки по удаленности от O, но для краткости будем говорить просто о сортировке по величине угла. Соединив точки в этом порядке (будем обозначать их $P_1, P_2, \ldots, P_n$, при этом $P_1 = P$), и соединив дополнительно P_n с P_1, мы получим замкнутую несамопересекающуюся ломаную, но ограниченный этой ломаной многоугольник может не быть выпуклым (см. рис. 2а). Тогда среди вершин $P_2, P_3, \ldots, P_n$ найдется хотя бы одна, скажем P_k, *вдавленная*, которая принадлежит треугольнику $P_{k-1}OP_{k+1}$ при $k < n$ и треугольнику $P_{n-1}OP_1$ при $k = n$ (рис. 2б). Вдавленную вершину можно исключить из дальнейшего рассмотрения, соединив напрямую P_{k-1} с P_{k+1}, или, соответственно, P_1 с P_{n-1}. Удалив все вдавленные вершины, мы получим требуемый многоугольник. Такова общая идея алгоритма. Задержимся на удалении вдавленных вершин.

[1] «Наглядно можно представлять себе дело так: в точках M вбиты гвозди, на которые натянута резинка, охватывающая их все, — эта резинка и будет выпуклой оболочкой множества гвоздей» [21]. Но в нашем понимании построение выпуклой оболочки предполагает еще перечисление вершин в порядке их обхода.

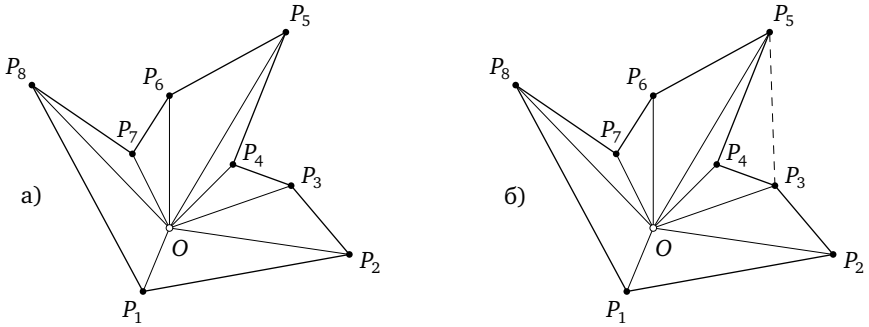


Рис. 2. а) Точки, упорядоченные по величине угла $\angle P_1OP_i$, $i = 1, 2, \ldots, n$; б) вершина P_4 — первая вдавленная вершина в последовательности $P_2, P_3, \ldots, P_8$.

Вдавленные вершины можно обнаружить просмотром точек $P_2, P_3, \ldots, P_n, P_1$: переходя от вершины P_i к вершине P_{i+1}, $i = 2, 3, \ldots, n-1$, можно сразу проверять, принадлежит ли P_i треугольнику $P_{i-1}OP_{i+1}$, а при переходе от P_n к P_1 проверять, принадлежит ли P_n треугольнику $P_{n-1}OP_1$. Если да, то P_i или соответственно P_n, удаляется, но после этого надо проверить, не окажется ли теперь вдавленной предыдущая из неудаленных вершин, — на рис. 2б видно, что после удаления P_4 вершина P_3 становится вдавленной. Возможно, что удаление одной вдавленной вершины повлечет удаление нескольких уже рассмотренных вершин, но вершина P_1 никогда не будет удалена. При $i < n-1$ после P_{i+1} мы рассматриваем P_{i+2} и вновь пытаемся освободиться от вдавленных вершин с меньшими номерами и т. д. Последний шаг — переход от P_n к P_1 и завершающая попытка освободиться от вдавленных вершин.

Затраты этапа построения точек P и O ограничены значением $c_1 n$, где c_1 — некоторая константа.

Если используется сортировка, сложность которой по числу сравнений есть $r(n)$, то в алгоритме Грэхема может потребоваться не более $c_2 r(n)$ операций для сортировки точек по величине угла, константа c_2 отражает затраты на сравнение двух углов и сравнение расстояний от O до двух данных точек.

Покажем, что описанный процесс удаления вдавленных вершин потребует затрат, не превосходящих по величине $c_3 n$, где c_3 — некоторая константа (в частности, учитывающая затраты на проверку принадлежности точки треугольнику). В самом деле, если переход от P_i к P_{i+1} сопровождается проверкой вдавленности некоторого числа v_i

вершин, то число удаленных при этом вершин равно $v_i - 1$. Но общее число удаленных вершин меньше n. Поэтому $\sum_{i=2}^{n}(v_i - 1) < n$, и, как следствие,

$$\sum_{i=2}^{n} v_i < 2n. \tag{3.1}$$

Это означает, что сложность $T_{\mathrm{G}}(n)$ алгоритма Грэхема по общему числу арифметических операций и сравнений не превосходит $c'r(n) + c''n$, где c' и c'' суть некоторые положительные константы. Сложность любой сортировки массивов длины n по числу сравнений не может быть меньше $n/2$, так как каждый элемент должен пройти хотя бы одно сравнение и в каждом сравнении участвуют два элемента. Имеем

$$T_{\mathrm{G}}(n) \leqslant c'r(n) + c''n = r(n)\left(c' + c''\frac{n}{r(n)}\right) \leqslant r(n)(c' + 2c''),$$

откуда $T_{\mathrm{G}}(n) = O(r(n))$. При этом у нас нет пока достаточных оснований для утверждения, что $T_{\mathrm{G}}(n) = \Theta(r(n))$, потому что нет, например, оснований утверждать, что после выбора P и O может действительно потребоваться $r(n)$ сравнений обсуждаемого типа (ведь мы можем еще выполнять арифметические операции; почему бы не предположить, что, прибегая к ним, можно существенно снизить число сравнений при сортировке), и мы пока можем утверждать только то, что $T_{\mathrm{G}}(n) = \Omega(n)$; эту тему мы отложим до § 29.

После того как точки упорядочены по величине угла, информацию об их координатах можно представить в виде двунаправленного списка, и тогда удаление вдавленных вершин не будет связано с какими-либо перемещениями координат точек, и, подходя формально, можно было бы затраты на перемещения в процессе удаления вдавленных вершин считать равными нулю. При менее формальном подходе эти затраты можно считать ограниченными сверху значением cn, где c — некоторая константа: переход от массива к двунаправленному списку и, равным образом, в силу (3.1), работа со ссылками во время удаления вдавленных вершин потребуют затрат, ограниченных величинами такого вида. В свою очередь, сложность любой сортировки по числу перемещений элементов не может быть меньше чем $n/2$ (так как не исключено, например, что изначальный порядок элементов является обратным к требуемому). Отсюда сложность алгоритма Грэхема по числу перемещений не превосходит произведения некоторой константы и сложности используемой сортировки по числу перемещений. Аналогично, для сложности по общему чис-

лу арифметических операций, сравнений и перемещений мы имеем оценку $O(s(n))$, где $s(n)$ — соответствующая сложность используемой сортировки.

Основываясь на эскизном описании алгоритма Грэхема, мы получили следующее.

Для любой сортировки массивов длины n, имеющей некоторую сложность $s(n)$ по общему числу сравнений и перемещений элементов, существует алгоритм построения выпуклой оболочки n точек, заданных массивом своих координат на плоскости, сложность которого по общему числу арифметических операций, сравнений и перемещений есть $O(s(n))$.

Пространственная сложность алгоритма Грэхема, очевидно, есть $O(n)$.

Пример 3.2. Пусть $G = (V, E)$ — ориентированный граф без кратных ребер и $v \in V$. *Вояжем* по G, выходящим из вершины v, будем называть любой путь, который

- начинается в вершине v,
- не проходит ни по одному из ребер дважды,
- завершается в вершине, из которой не выходит ни одного непройденного ребра

(вояж не обязательно охватывает все ребра G). Примером выходящего из вершины 1 вояжа в изображенном на рис. 3 графе служит $(1, 2, 2, 3, 1, 4)$.

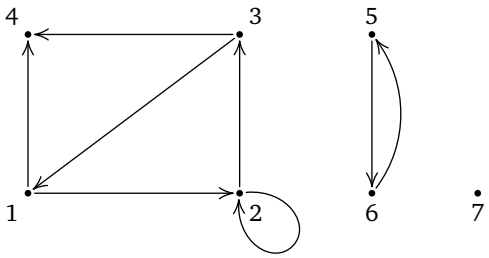


Рис. 3. Ориентированный граф.

Построение какого-то одного выходящего из данной вершины v вояжа может быть выполнено произвольным блужданием по непройденным ребрам графа, завершающимся в вершине, из которой не выходит непройденных ребер. Систематизация блужданий может со-

стоять в том, что, попав в некоторую вершину, мы выбираем для продолжения пути ребро, ведущее в вершину с наименьшим доступным номером.

Определяемый этим алгоритмом вояж может захватить и все ребра — например, когда граф имеет вид кольца (на рис. 4 изображен такой граф, для которого $|E| = |V| = 5$). Входом алгоритма является граф G и вершина v. Если мы рассматриваем $|E|$ как размер входа, то для сложности любого алгоритма построения вояжа обязательно имеет место нижняя оценка $\Omega(|E|)$ — ввиду, например, того, что для любого фиксированного $|E|$ существует граф, имеющий вид кольца. Возникает вопрос, можно ли обосновать верхнюю оценку $O(|E|)$? Если да, то в итоге это дало бы оценку $\Theta(|E|)$.

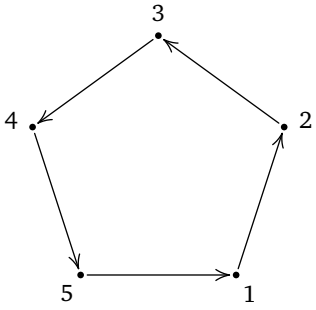


Рис. 4. Граф в виде кольца (любой вояж охватывает все ребра).

Для представления графов, не имеющих кратных ребер, часто используются *матрицы смежности*. Матрица смежности — квадратная матрица порядка $|V|$, состоящая из нулей и единиц (фактически, булева матрица), в которой элемент с индексами i, j равен единице, если и только если из i-й вершины выходит ребро в j-ю вершину. Такой способ представления действительно удобен во многих задачах. Но когда речь идет о блужданиях по графу, при которых однажды пройденное ребро изымается из дальнейшего рассмотрения, более удобным оказывается представление графа в виде массива $a_1, a_2, \ldots, a_{|V|}$ списков смежных вершин. В списке a_i, $i = 1, 2, \ldots, |V|$, в порядке возрастания содержатся номера вершин, в которые входят ребра, выходящие из вершины с номером i. Для графа, приведенного на рис. 3, имеем

$$a_1 = (2, 4), \quad a_2 = (2, 3), \quad a_3 = (1, 4), \quad a_4 = (), \quad a_5 = (6), \quad a_6 = (5), \quad a_7 = ();$$

для графа в виде кольца —

$$a_1 = (2), \quad a_2 = (3), \quad \ldots, \quad a_{|V|-1} = (|V|), \quad a_{|V|} = (1)$$

(в этом графе $|V| = |E|$). Представление в виде массива списков смежных вершин возможно и для графов, имеющих кратные ребра; в этом случае номера вершин в каждом списке располагаются в порядке неубывания.

Начав вояж из вершины с номером v, мы смотрим, не пуст ли список a_v: если он пуст, то вояж закончен. Если же список a_v не пуст, переносим первый его элемент (пусть это будет, например, i) в список вершин, через которые шаг за шагом проходит вояж; затем рассматриваем список a_i, если он пуст, то вояж закончен, иначе повторяем все то же самое, что проделывали в предыдущей вершине, и т. д.:

$l := cons(v, nil)$; $i := v$;
`while not` $null(a_i)$ `do` $i := car(a_i)$; $l := cons(i, l)$; $a_i := cdr(a_i)$ `od`

Мы здесь использовали операции над списками, применяемые в языке Лисп: *car* — первый элемент списка, *cdr* — список после удаления первого его элемента, *cons* — вставка элемента в начало списка, *null* — предикат проверки пустоты списка, *nil* — обозначение пустого списка.

Затраты, связанные с удлинением пути на один шаг и с проверкой, не завершен ли вояж, ограничены сверху константой (эти затраты суть четыре операции над списками), длина всего пути не превосходит $|E|$, и затраты во всех случаях не превосходят произведения некоторой константы на $|E|$.

Список l содержит в обратном порядке все вершины, через которые последовательно проходит вояж. Если желателен прямой порядок, то надо перевернуть l:

$l' := nil$;
`while not` $null(l)$ `do` $l' := cons(car(l), l')$; $l := cdr(l)$ `od`

Построение списка l' потребует некоторых дополнительных операций над списками, число этих операций не превзойдет произведения некоторой константы c на длину списка l, но эта длина не превосходит $|E|$, следовательно, число операций на этом этапе не превзойдет $c|E|$.

Естественно считать пространственные затраты на хранение списка с числовыми элементами пропорциональными длине списка. В случае, например, графа в виде кольца каждый из списков l и l' имеет длину $|E| + 1$.

Обозначим буквами VL описанный алгоритм построения вояжа, предполагая, что исходный ориентированный граф задается массивом списков смежных вершин. Из сказанного следует, что для временных затрат алгоритма VL мы имеем

$$C^T_{\mathrm{VL}}(G, v) \leqslant c|E|, \tag{3.2}$$

где c — некоторая константа. Это неравенство выполнено для любого имеющего $|E|$ ребер графа. Поэтому если значение $|E|$ принято за размер входа (напомним, что сам вход — это граф и выделенная в нем вершина), то из (3.2) будет следовать $T_{\text{VL}}(|E|) \leqslant c|E|$, откуда

$$T_{\text{VL}}(|E|) = O(|E|). \tag{3.3}$$

Вместе с указанной ранее оценкой $\Omega(|E|)$ оценка (3.3) дает

$$T_{\text{VL}}(|E|) = \Theta(|E|).$$

Нетрудно вывести аналогичную оценку и для пространственной сложности.

Если массивы списков смежных вершин используются в качестве представления ориентированных графов, то построение начинающегося с заданной вершины v вояжа в графе $G = (V, E)$ возможно с помощью алгоритма, который при выборе числа $|E|$ ребер в качестве размера входа имеет сложность $\Theta(|E|)$ по числу операций над списками. Эта же оценка имеет место для пространственной сложности, если считать, что пространственные затраты на хранение списка с числовыми элементами пропорциональны длине этого списка.

Если бы граф представлялся матрицей смежности, то мы для используемого алгоритма не смогли бы получить оценку сложности $\Theta(|E|)$, так как при блуждании по графу вход в каждую вершину сопровождается проверкой, есть ли непройденное до сих пор ребро, выходящее из этой вершины. Например, для графа в виде кольца при $|E| = |V| = n$ мы имеем матрицу смежности порядка n:

$$\begin{pmatrix} 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & 1 \\ 1 & 0 & 0 & \dots & 0 \end{pmatrix}.$$

Начиная вояж из первой вершины, мы потратим $2 + 3 + \dots + n + 1 = \Omega(n^2)$ сравнений элементов матрицы с единицей при поиске первых подходящих вершин.

Разумеется, базовые операции над списками и операции сравнения элементов матрицы смежности с единицей различаются по затратности (сравнение с единицей — это очень дешевая операция). Но какими бы ни были положительные c' и c'', отражающие затраты на операцию сравнения с 1 и, соответственно, на операцию над списком, неравенство $c'|E|^2 > c''|E|$ будет иметь место для всех достаточно

больших $|E|$. Константы, скрытые в оценках $\Omega(|E|^2)$ и $O(|E|)$, найденных для сложностей алгоритма при использовании двух разных способов представления графа, различаются между собой. Но тем не менее мы можем сравнивать рост первой и второй сложностей на основе этих оценок. Этот пример показателен и в том отношении, что «дешевых» операций может быть слишком много (их число может слишком быстро расти при увеличении размера входа) и их игнорирование может дать неправильное представление о реальной временной сложности алгоритма.

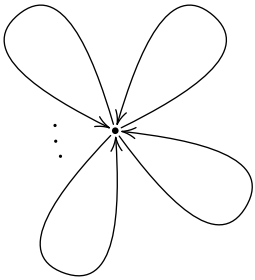


Рис. 5. Граф с одной вершиной и произвольным заданным числом ребер.

Если рассматривать два параметра $|V|, |E|$ размера входа, то сложностью алгоритма VL будет функция $T'_{\mathrm{VL}}(|V|, |E|)$ двух переменных. Для того чтобы устанавливать для нее асимптотические оценки, эта функция должна быть определена для всех (или, по крайней мере, всех достаточно больших) натуральных значений $|V|, |L|$. Это значит, что для всех таких $|V|, |E|$ должен существовать граф, имеющий $|V|$ вершин и $|E|$ ребер. Проблема будет снята, если, например, допустить к рассмотрению и такие ориентированные графы, которые имеют кратные ребра. Оценка $T'_{\mathrm{VL}}(|V|, |E|) = O(|E|)$ будет, очевидно, выполняться. Можно обосновать и оценку $T'_{\mathrm{VL}}(|V|, |E|) = \Omega(|E|)$: достаточно рассмотреть имеющий $|E|$ ребер граф в виде цветка (рис. 5), добавив к нему $|V| - 1$ изолированных вершин (подобных вершине 7 на рис. 3) и взяв в качестве v вершину в центре «цветка». В итоге $T'_{\mathrm{VL}}(|V|, |E|) = \Theta(|E|)$.

§ 4. Длина числа как возможный размер входа

Часто, хотя и не всегда, для алгоритмов целочисленной арифметики, входом которых является целое неотрицательное число n, размером входа выбирают не само n, а его *битовую длину*, или, иными словами, количество $\lambda(n)$ цифр в его двоичной записи:

$$\lambda(n) = \begin{cases} 1, & \text{если } n = 0, \\ \lfloor \log_2 n \rfloor + 1, & \text{если } n > 0. \end{cases}$$

Выражение $\lfloor \log_2 n \rfloor + 1$ во второй строчке этого определения можно заменить на $\lceil \log_2 (n+1) \rceil$ — см. задачу 2.

Пример 4.1. Вернемся к алгоритму пробных делений (пример 1.2). Наряду с его уже рассмотренной сложностью $T_{\mathrm{TD}}(n)$ введем еще одну сложность (также по числу делений) $T^*_{\mathrm{TD}}(m)$, $m = \lambda(n)$. Легко обнаружить, что эта новая функция ведет себя не так, как функция $T_{\mathrm{TD}}(n)$, для которой характерны резкие скачки. Для начальных значений m имеем

m:	2	3	4	5	6	7
n:	2—3	4—7	8—15	16—31	32—63	64—127
$\hat{n}$:	3	7	13	31	61	127
$T^*_{\mathrm{TD}}(m)$:	1	1	2	4	6	10

(в строке m указывается битовая длина входа; в строке n указывается диапазон, в котором располагаются значения n, битовая длина m которых равна числу в предыдущей строке; в строке $\hat{n}$ указывается одно из тех чисел этого диапазона, на которых достигается максимум числа делений; в последней строке указано значение сложности $T^*_{\mathrm{TD}}(m)$). В поведении $T^*_{\mathrm{TD}}(m)$ не видно резких скачков. Переход от размера $\|n\| = n$ к размеру $\|n\|^* = m = \lambda(n)$ влечет более основательное преобразование функции $T(n)$, чем простая замена переменной. Теперь одним и тем же размером m обладают несколько входов n, затраты для которых, вообще говоря, различны, и, в соответствии с определением сложности, мы должны брать максимум этих затрат. Данным размером m обладают все целые n такие, что

$$2^{m-1} \leqslant n < 2^m. \tag{4.1}$$

Согласно постулату Бертрана, при $m > 1$ в таком диапазоне обязательно встретится простое число.

Постулат Бертрана.[1] *При любом целом $N > 1$ имеется простое число, принадлежащее интервалу $(N, 2N)$.*

С помощью постулата Бертрана мы можем доказать, что $T^*_{\mathrm{TD}}(m) = \Theta(2^{m/2})$. В самом деле, обозначим через p_m наибольшее простое число битовой длины m. Количество делений, требующееся алгоритму для работы с p_m, будет равно $\lfloor\sqrt{p_m}\rfloor - 1$. В силу (4.1) имеем $\lfloor 2^{(m-1)/2}\rfloor - 1 \leqslant \lfloor\sqrt{p_m}\rfloor - 1$. Таким образом,

$$T^*_{\mathrm{TD}}(m) \geqslant \lfloor 2^{(m-1)/2}\rfloor - 1 > \frac{1}{\sqrt{2}} 2^{m/2} - 2 = \left(\frac{1}{\sqrt{2}} - \frac{1}{2^{m/2-1}}\right) 2^{m/2},$$

[1] Сформулирован Ж. Бертраном в 1845 г. и доказан П. Л. Чебышевым в 1852 г. Обсуждение доказательства выходит за рамки этого курса, доказательство Чебышева см. в [38] (статья «О простых числах»). Доказательства, использующее лишь элементарные средства, можно найти в [35] (приложение «Доказательство постулата Бертрана (теоремы Чебышева)») и в [11, § 5].

и при $m \geqslant 4$ имеем

$$T^*_{\mathrm{TD}}(m) > \Big(\frac{1}{\sqrt{2}} - \frac{1}{2}\Big)2^{m/2}.$$

В то же время, в силу (4.1) очевидно, что для всех m

$$T^*_{\mathrm{TD}}(m) < 2^{m/2} - 1 < 2^{m/2}.$$

Отсюда следует требуемое.

Исследуем общий вопрос о возможности перехода от оценок $T^*_A(m)$ к оценкам $T_A(n)$ и наоборот для какого-либо алгоритма A, считая, что эти две сложности соответствуют одному и тому же способу учета затрат, но разным размерам входа — $\|\cdot\|$ и $\|\cdot\|^*$, принимающим значения в $\mathbb{N}^+$, причем если $\|x\| = n$ для некоторого входа x, то $\|x\|^* = m = \lambda(n) = \lfloor \log_2 n \rfloor + 1$. В таком случае, очевидно, выполнено

$$T^*_A(m) = \max_{2^{m-1} \leqslant n < 2^m} T_A(n). \tag{4.2}$$

Ясно, что $T_A(n)$ несет более полную информацию о рассматриваемом алгоритме A, чем $T^*_A(m)$: значения $T^*_A(m)$, $m = 1, 2, \ldots$, образуют подпоследовательность последовательности $T_A(n)$, $n = 1, 2, \ldots$ Поэтому естественно, что переход от оценок для $T^*_A(m)$ к оценкам для $T_A(n)$ приводит к более грубому результату, чем тот, который может быть получен при изначальном рассмотрении размера $\|x\| = n$. Особенно это касается нижних оценок (пример 4.1 прямо указывает на это).

Лемма 4.1. *Пусть $f(x)$ — неубывающая функция вещественной переменной. Тогда если $T^*_A(m) \leqslant f(m)$, то $T_A(n) \leqslant f(\log_2 n + 1)$.*

Доказательство. Пусть m и n фиксированы, $2^{m-1} \leqslant n < 2^m$, и пусть значение $\hat{n}$ таково, что

$$2^{m-1} \leqslant \hat{n} < 2^m, \tag{4.3}$$

и при этом

$$T_A(\hat{n}) = T^*_A(m). \tag{4.4}$$

Используем неубывание $f(x)$:

$$T_A(n) \leqslant T_A(\hat{n}) = T^*_A(m) \leqslant f(m) = f(\lfloor \log_2 n \rfloor + 1) \leqslant f(\log_2 n + 1). \qquad \square$$

Лемма 4.2. *Пусть $g(x)$ — неубывающая функция вещественной переменной. Тогда*

(i) *если $T_A(n) \leqslant g(n)$, то $T^*_A(m) \leqslant g(2^m)$,*

(ii) *если $T_A(n) \geqslant g(n)$, то $T^*_A(m) \geqslant g(2^{m-1})$.*

Доказательство. Пусть m фиксировано, $2^{m-1} \leqslant n < 2^m$, и пусть значение $\hat{n}$ таково, что выполнены (4.3) и (4.4). Используем неубывание $g(x)$:

(i) $T_A^*(m) = T_A(\hat{n}) \leqslant g(\hat{n}) \leqslant g(2^m)$,

(ii) $T_A^*(m) = T_A(\hat{n}) \geqslant g(\hat{n}) \geqslant g(2^{m-1})$. □

Утверждения следующих двух теорем непосредственно следуют из доказанных лемм. (При рассмотрении асимптотической оценки, содержащей некоторую переменную, подразумевается, что значение этой переменной стремится к бесконечности.)

Теорема 4.1. *Пусть $f(x)$ — неубывающая функция вещественной переменной. Тогда если $T_A^*(m) = O(f(m))$, то $T_A(n) = O(f(\log_2 n + 1))$; как следствие, при $f(x+1) = O(f(x))$ имеем $T_A(n) = O(f(\log_2 n))$.*

Теорема 4.2. *Пусть $g(x)$ — неубывающая функция вещественной переменной. Тогда*

(i) *если $T_A(n) = O(g(n))$, то $T_A^*(m) = O(g(2^m))$,*

(ii) *если $T_A(n) = \Omega(g(n))$, то $T_A^*(m) = \Omega(g(2^{m-1}))$; как следствие, при $g\left(\frac{x}{2}\right) = \Omega(g(x))$ имеем $T_A^*(m) = \Omega(g(2^m))$.*

Существуют функции, для которых условие $f(x+1) = O(f(x))$ или $g\left(\frac{x}{2}\right) = \Omega(g(x))$ не выполнено — например, $f(x) = 2^{x^2}$, $g(x) = 2^x$.

Для рассмотренных в примере 4.1 сложностей $T_{\text{TD}}(n)$, $T_{\text{TD}}^*(m)$ ситуация выглядит следующим образом. Функция $f(x) = 2^{x/2}$ является возрастающей, и по теореме 4.1 из $T_{\text{TD}}^*(m) = O(2^{m/2})$ следует $T_{\text{TD}}(n) = O(2^{(\log_2 n+1)/2}) = O(\sqrt{n})$. Рассматривая возрастающую функцию $g(x) = \sqrt{x}$, мы можем, применив теорему 4.2(i), вывести из $T_{\text{TD}}(n) = O(\sqrt{n})$ оценку $T_{\text{TD}}^*(m) = O(\sqrt{2^m}) = O(2^{m/2})$.

Получить из оценки $T_{\text{TD}}^*(m) = \Omega(2^{m/2})$ оценку $T_{\text{TD}}(n) = \Omega(\sqrt{n})$ мы не можем, так как последняя оценка не верна; это не противоречит доказанным утверждениям.

Пример 4.2. Идея бинарного алгоритма возведения a в целую неотрицательную степень n, называемого также алгоритмом повторного возведения в квадрат (мы будем обозначать его буквами RS, от английского названия алгоритма repeated squaring — повторное возведение в квадрат), состоит в том, что если двоичная запись n есть $\beta_k \ldots \beta_1 \beta_0$, то вычисление a^n может быть сведено к вычислению последовательности значений $q_i = a^{(2^i)}$, $i = 0, 1, \ldots, k$ (каждый следующий элемент последовательности получаем возведением в квадрат преды-

дущего), и подсчету произведения u тех q_i, для которых $\beta_i = 1$:

```
q := a; u := 1;
for i = 0 to k − 1 do
   if β_i = 1 then u := u · q fi;
   q := q^2
od
u := u · q
```

Например, $a^{11} = a^8 \cdot a^2 \cdot a$ (5 умножений), так как $(11)_{10} = (1011)_2$.

Займемся сложностью по числу умножений этого алгоритма. Используя $\lambda(n)$ для обозначения битовой длины n и обозначение $\lambda^*(n)$ для числа единиц в двоичной записи n, мы легко получаем следующее.

Если рассматривать n как размер входа бинарного алгоритма вычисления a^n, $n \in \mathbb{N}^+$, то сложность $T_{\mathrm{RS}}(n)$ по числу умножений для этого алгоритма равна $\lambda(n) + \lambda^*(n) - 2$. Если в качестве размера входа рассматривать $m = \lambda(n)$, то сложность $T^*_{\mathrm{RS}}(m)$ по числу умножений равна $2m - 2$.

Для $T^*_{\mathrm{RS}}(m)$ и $T_{\mathrm{RS}}(n)$ имеем очевидные асимптотические оценки

$$T^*_{\mathrm{RS}}(m) = \Theta(m), \quad T_{\mathrm{RS}}(n) = \Theta(\log n).$$

Заметим, что мы не можем вывести из $T^*_{\mathrm{RS}}(m) = 2m - 2$ равенство $T_{\mathrm{RS}}(n) = \lambda(n) + \lambda^*(n) - 2$, хотя обратный вывод очевиден.

В рассмотренном алгоритме предполагается, что показатель степени n задан как массив двоичных цифр. Если показатель n задан как число, то его двоичные цифры $\beta_0, \beta_1, \ldots$ могут быть получены одна за другой нахождением соответствующих частных и остатков от деления на 2. Это не изменит оценок $\Theta(\log n)$, $\Theta(m)$ как для общего числа операций, так и для числа мультипликативных операций. Но при этом бинарный алгоритм возведения в степень может быть использован, например, для вычисления A^n, где A — матрица большого порядка и т. д. В этом смысле те операции, которые подразумеваются в командах $u := u \cdot q$ и $q := q^2$, естественно подсчитывать отдельно и именно их считать составляющими главные затраты алгоритма.

Задачу вычисления a^n в некотором множестве с ассоциативным умножением (т. е. в полугруппе) часто формулируют как задачу об *аддитивных цепочках* для n, т. е. о наборах целых чисел $n_1 < n_2 < \ldots < n_k$, в которых

- $n_1 = 1$, $n_k = n$;
- для каждого i, $1 < i \leqslant k$, найдутся s, t такие, что $1 \leqslant t \leqslant s < i$ и $n_t + n_s = n_i$.

Каждая аддитивная цепочка для n задает способ вычисления a^n. Например, аддитивная цепочка $1, 2, 3, 4, 8, 11$ для 11 задает тот способ, который соответствует бинарному алгоритму возведения в степень. Задача быстрого возведения в степень n с помощью умножений — это фактически задача построения короткой аддитивной цепочки для n.

Обозначение $l(n)$ закреплено за длиной самой короткой аддитивной цепочки для n. Бинарный алгоритм возведения в степень использует свою специфическую аддитивную цепочку (назовем ее бинарной). Бинарная цепочка не для всех n имеет длину $l(n)$ (см. задачу 19). Но, как мы выясним несколько позже, длина бинарной цепочки и $l(n)$ имеют одинаковый порядок. Помимо этого бинарные цепочки легко и быстро строятся, чего в общем случае нельзя сказать об аддитивных цепочках длины $l(n)$.

Некоторые предположения относительно функции $l(n)$ до настоящего времени не доказаны, хотя и подтверждены проверкой для многих n. К числу таковых относится, например, гипотеза Шольца—Брауэра: $l(2^n-1) \leqslant n-1+l(n)$ для всех $n>0$. [1]

Задачи

1. Для любого $n \in \mathbb{Z}$ выполнено $n = \left\lfloor \frac{n}{2} \right\rfloor + \left\lceil \frac{n}{2} \right\rceil$, для любого $a \in \mathbb{R}$ выполнено $\lceil a \rceil = -\lfloor -a \rfloor$.

2. Для любых $k, n \in \mathbb{N}^+$, $k>1$, выполнено $\lfloor \log_k n \rfloor + 1 = \lceil \log_k (n+1) \rceil$.

Указание. Пусть $m \in \mathbb{N}^+$ таково, что $k^{m-1} \leqslant n < k^m$, тогда $k^{m-1} < n+1 \leqslant k^m$. Прологарифмировать по основанию k обе системы неравенств.

3. Предположим, что в нашем распоряжении нет операции обмена $a \leftrightarrow b$ и мы заменяем ее тремя присваиваниями:

$$c := a; \quad a := b; \quad b := c;$$

чему в этом случае равны сложности первого и второго варианта сортировки простыми вставками по суммарному числу сравнений и присваиваний?

4. Пусть полином $f(n) = a_m n^m + \ldots + a_1 n + a_0$ имеет ненулевой старший коэффициент a_m. Тогда

а) $f(n) = O(n^k) \Leftrightarrow k \geqslant m$,

б) $f(n) = \Omega(n^k) \Leftrightarrow k \leqslant m$,

в) $f(n) = \Theta(n^k) \Leftrightarrow k = m$, при $k = m$ также выполнено $f(n) \sim a_m n^m$.

[1] Подробнее об аддитивных цепочках см. в [19, разд. 4.6.3].

5. Пусть $P(n)$ — количество простых множителей целого числа $n > 1$ с учетом кратности. Имеет место точная оценка $P(n) = O(\log n)$.

6. (Продолжение предыдущей задачи.) Пусть

$$P^*(m) = \max_{2^{m-1} \leqslant n < 2^m} P(n),$$

$m = 2, 3, \ldots$ Верно ли, что $P^*(m) = \Theta(m)$?

7. Указать все вещественные значения δ, при которых справедлива оценка

а) $T_{\mathrm{TD}}(n) = O(n^\delta)$,

б) $T_{\mathrm{TD}}(n) = \Omega(n^\delta)$,

в) $T_{\mathrm{TD}}(n) = \Theta(n^\delta)$,

где $T_{\mathrm{TD}}(n)$ — сложность алгоритма пробных делений (пример 1.2).

8. Железнодорожный сортировочный узел устроен так, как показано на рис. 6. На правой стороне собрано некоторое число вагонов

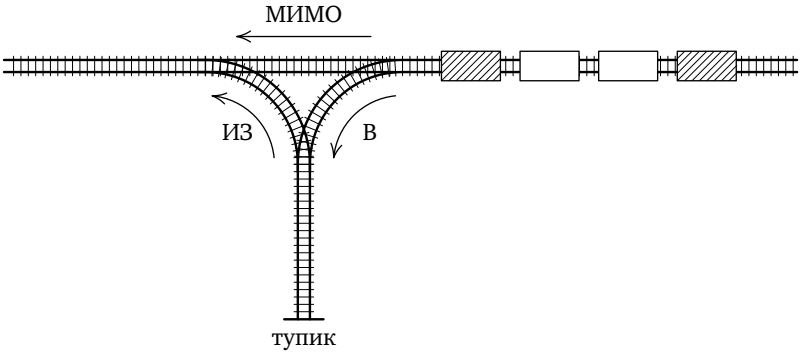


Рис. 6. Сортировочный узел.

двух типов (на рис. 6 — черные и белые), по n штук каждого типа. Тупик может вместить все $2n$ вагонов. Требуется, пользуясь тремя сортировочными операциями В, ИЗ, МИМО, собрать вагоны на левой стороне так, чтобы типы вагонов чередовались. Указать такой алгоритм решения этой задачи, сложность которого по числу сортировочных операций при рассмотрении n в качестве размера входа равнялась бы $3n - 1$.

9. Хорошо известно, что мультипликативная сложность метода Гаусса решения системы n линейных уравнений с n неизвестными допускает оценки

$$1)\ O(n^3), \quad 2)\ \frac{1}{3}n^3 + O(n^2), \quad 3)\ \Theta(n^3).$$

а) Из какой оценки (указать номер) следуют две остальные?

б) Можно ли из приведенных оценок выбрать такую, которая является следствием любой из остальных?

в) Является ли оценка $\Omega(n^3)$ следствием какой-либо из оценок 1, 2, 3?

10. Для сложности $T_{\mathrm{QS}}(n)$ быстрой сортировки выполняется оценка $T_{\mathrm{QS}}(n) = \Theta(n^2)$. (Обозначение QS происходит от английского названия быстрой сортировки — quick sort.)

Указание. Оценка $T_{\mathrm{QS}}(n) = \Omega(n^2)$ устанавливается предъявлением примера. Неравенство $T_{\mathrm{QS}}(n) < n^2$ можно доказать индукцией по n, используя то, что при фиксированном $n \in \mathbb{N}^+$ квадратичная функция $(m-1)^2 + (n-m)^2$ от m принимает на отрезке $[1, n]$ свое максимальное значение в одном из концов этого отрезка.

11. Алгоритм, использующий только аддитивные операции и сравнения целых чисел для проверки того, является ли данное целое положительное n точным квадратом, может быть основан на вычислении последовательности значений $1, 1+3, 1+3+5, \ldots$ до появления в ней первого большего или равного n элемента. Сложность по числу аддитивных операций и операций сравнения этого алгоритма (назовем его sq_1) допускает оценку $\Theta(\sqrt{n})$. Сохранится ли эта оценка, если дополнительно использовать операцию нахождения целой части половины числа (считается, что затратность этой операции такая же, как у аддитивных операций) для того, чтобы предварительно выяснять, на какую максимальную степень двойки — четную или нечетную — делится n (алгоритм sq_2)?

12. (Продолжение предыдущей задачи.) Пусть $T_{\mathrm{sq}_1}(n)$, $T_{\mathrm{sq}_2}(n)$ — сложности, соответственно, первого и второго вариантов рассмотренного алгоритма и

$$T^*_{\mathrm{sq}_1}(m) = \max_{2^{m-1} \leqslant n < 2^m} T_{\mathrm{sq}_1}(n), \quad T^*_{\mathrm{sq}_2}(m) = \max_{2^{m-1} \leqslant n < 2^m} T_{\mathrm{sq}_2}(n),$$

$m = 2, 3, \ldots$

а) Как связаны значения функций $T^*_{\mathrm{sq}_1}(m)$ и $T^*_{\mathrm{sq}_2}(m)$?

б) Имеются ли среди оценок $\Theta(m)$, $O(\log m)$, $\Theta(2^{m/2})$, $O(2^m)$ такие, которые верны для $T^*_{\mathrm{sq}_2}(m)$, и если да, то какие именно?

13. Изменим в алгоритме Грэхема (пример 3.1) этап удаления вдавленных вершин: будем обходить — возможно, многократно — против часовой стрелки вершины построенной несамопересекающейся ломаной и удалять вдавленные вершины до тех пор, пока при очередном обходе не окажется безуспешной попытка найти хотя бы

одну вдавленную вершину. Сложность этого варианта рассматриваемого этапа алгоритма Грэхема есть $\Omega(n^2)$, где n — начальное число вершин ломаной (в то время как сложность этого этапа в алгоритме Грэхема есть $O(n)$).

14. Рассмотрим в главных чертах алгоритм Шеймоса—Хоя[1] построения пересечения $P \cap Q$ двух выпуклых многоугольников P и Q содержащих соответственно l и m вершин. Считаем, что многоугольники задаются массивами $P_1, P_2, \ldots, P_l$ и $Q_1, Q_2, \ldots, Q_m$ координат своих последовательных вершин.

Упорядочим множество всех абсцисс обоих многоугольников по возрастанию (для простоты считаем, что абсциссы попарно различны, хотя это и несущественно), в результате получим абсциссы $a_1 < a_2 < \ldots < a_{l+m}$ (рис. 7). Теперь для каждого $i = 1, 2, \ldots, l+m-1$

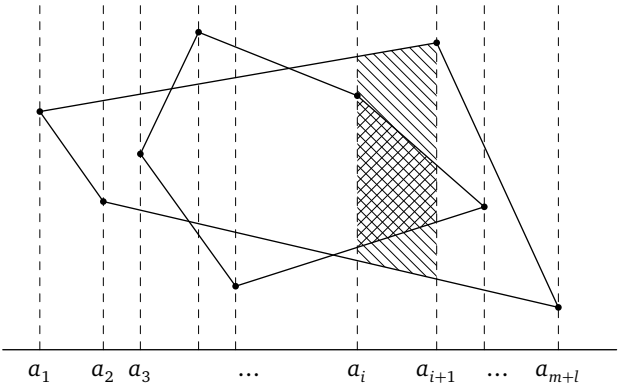


Рис. 7. Алгоритм Шеймоса—Хоя ($l = 5$, $m = 4$).

строим пересечение трапеций, вырезаемых прямыми $x = a_i, x = a_{i+1}$ в заданных многоугольниках. (В вырожденном случае такая трапеция превращается в треугольник.) Из получившихся пересечений трапеций можно собрать $P \cap Q$, удаляя лишние вершины.

Привлекая дополнительно те или иные структуры данных (массивы, списки, ...) и уточняя по мере надобности какие-то этапы алгоритма, добиться того, чтобы в итоге алгоритм (обозначим его буквами SH) имел следующие сложностные характеристики. Если размер входа — это $r = l + m$, то $T_{\mathrm{SH}}(r) = \Theta(r)$; если размер входа — это $s = \max\{l, m\}$, то $T'_{\mathrm{SH}}(s) = \Theta(s)$; если размер входа имеет два параметра

[1] См. [30, гл. 7].

l и m, то $T''_{\text{SH}}(l,m)=\Theta(l+m)$ — мы рассматриваем операции сравнения и перемещения элементов, арифметические операции, как мультипликативные, так и аддитивные, все вместе. Для пространственной сложности — $S_{\text{SH}}(r)$, $S'_{\text{SH}}(s)$ или $S''_{\text{SH}}(l,m)$ в зависимости от выбора размера входа — получаем те же оценки $\Theta(r)$, $\Theta(s)$ или $\Theta(l+m)$.

Указание. Если исходные многоугольники представить, например, двунаправленными списками координат последовательных вершин, то список абсцисс $(a_1, a_2, \ldots, a_{l+m})$, $a_1 < a_2 < \ldots < a_{l+m}$, можно получить с временными затратами, не превосходящими $c_0(l+m)$.

При определении пространственной сложности можно заметить, что общее число вершин искомого пересечения не превосходит $3(l+m)$ — это следует непосредственно из самого алгоритма Шеймоса—Хоя: при построении пересечения двух трапеций может возникнуть не более двух дополнительных вершин.

15. Расширить алгоритм построения вояжа в ориентированном графе $G=(V,E)$ с выделенной начальной вершиной v (пример 3.2) так, чтобы помимо вояжа алгоритм выдавал также информацию о том, охватил ли вояж все ребра графа. Размер входа имеет два параметра $m=|V|$, $n=|E|$. Для графов, не содержащих изолированных вершин, т. е. вершин, подобных вершине 7 на рис. 3, сложность расширенного алгоритма должна быть $O(n)$.

16. Задача распознавания существования и фактического построения эйлерова цикла в данном графе для ориентированного случая выглядит так: выяснить, имеется ли в данном ориентированном графе цикл, проходящий по всем ребрам графа по одному разу, и если существует, то построить этот цикл. Воспользовавшись рассмотренным алгоритмом построения вояжа (пример 3.2), дать алгоритм решения этой задачи. Размер входа имеет два параметра $m=|V|$, $n=|E|$. Для графов, не содержащих изолированных вершин, сложность предлагаемого алгоритма должна быть $O(n)$.

17. Путник столкнулся со стеной, простирающейся бесконечно в обе стороны. Имеется дверь в этой стене, но путник не знает ни расстояния до двери, ни направления к ней. Предложить позволяющий добраться до двери алгоритм, сложность которого допускает оценку $O(n)$, где n — число шагов (неизвестное путнику), изначально отделяющее путника от двери.

Указание. Сумма $s_k=1+q+\ldots+q^k$, $q>1$, есть величина порядка q^k, т. е. $s_k=\Theta(q^k)$.

18. Будем считать затратностью любого построения на плоскости с помощью циркуля и линейки количество линий (окружностей или

прямых), проводимых в ходе построения. Рассмотрим задачу построения отрезка, длина которого равна $1/n$ от длины исходного отрезка, считая n размером входа. Предложить для этой задачи такой алгоритм решения, сложность которого допускает оценку $O(\log n)$. [1]

19. При $n = 15$ возможно вычисление a^n с меньшим числом умножений, чем требуется бинарному алгоритму возведения в степень.

20. Пусть двоичная запись числа $n \in \mathbb{N}$ есть $\beta_k \ldots \beta_1 \beta_0$. Алгоритм

```
u := 1;
for i = k downto 0 do
    u := u · u;
    if β_i = 1 then u := u · a fi
od
```

вычисляет a^n. Сравнить сложность этого алгоритма по числу умножений с аналогичной сложностью бинарного алгоритма возведения в степень, рассмотренного в примере 4.2, при использовании n или соответственно $m = \lambda(n)$ в качестве размера входа. (Описанный в этой задаче алгоритм является еще одним вариантом бинарного алгоритма возведения в степень.)

21. Для функции $l(n)$, определенной в конце § 4, выполнено $l(ab) \leqslant \leqslant l(a) + l(b) - 1$ для всех $a, b \in \mathbb{N}^+$.

22. (Продолжение предыдущей задачи.) Можно ли утверждать, что $l(ab) = l(a) + l(b) - 1$ для всех $a, b \in \mathbb{N}^+$?

[1] Разбор задач на построение с точки зрения их сложности содержится, например, в [11, § 15].

Глава 2

Сложность в среднем

§ 5. Понятие сложности в среднем

До сих пор мы исследовали сложность в худшем случае, теперь обратимся к сложности в среднем. Мы ограничимся ситуацией, когда при каждом фиксированном значении s размера входа сами соответствующие входы образуют конечное множество $X_s = \{x\colon \|x\| = s\}$. Будем предполагать, что каждому $x \in X_s$ приписана некоторая вероятность $\mathsf{P}_s(x)$:

$$0 \leqslant \mathsf{P}_s(x) \leqslant 1,$$

и при этом

$$\sum_{x \in X_s} \mathsf{P}_s(x) = 1.$$

Таким образом, при каждом допустимом значении s размера входа множество X_s превращается в вероятностное пространство; по умолчанию считается, что вероятность распределена равномерно:

$$\mathsf{P}_s(x) = \frac{1}{N_s}, \tag{5.1}$$

где N_s — количество элементов множества X_s. Функции от s

$$\sum_{x \in X_s} C_A^T(x)\mathsf{P}_s(x) \quad \text{и} \quad \sum_{x \in X_s} C_A^S(x)\mathsf{P}_s(x) \tag{5.2}$$

называются временной и, соответственно, пространственной сложностью в среднем алгоритма A. Более подробно:

Определение 5.1. Пусть при любом допустимом значении s множество X_s всех входов размера s является вероятностным пространством, в силу чего временные и пространственные затраты алгоритма A для входов x (т. е. $C_A^T(x)$ и $C_A^S(x)$) размера s являются случайными величинами на X_s. *Сложностью в среднем* называется математическое ожидание (первая или вторая сумма из (5.2)) соответствующей случайной величины.

Сложность в среднем может принимать и нецелые значения, даже если функция затрат целочисленна.

Для временной и пространственной сложности в среднем алгоритма A мы будем использовать обозначения $\overline{T}_A(\cdot)$, $\overline{S}_A(\cdot)$.

Теорема 5.1. *Для любого алгоритма A при любом распределении вероятностей на множестве X_s, где s — некоторое допустимое значение размера $\|\cdot\|$, сложность в среднем не превосходит сложности в худшем случае:*

$$\overline{T}_A(s) \leqslant T_A(s), \quad \overline{S}_A(s) \leqslant S_A(s).$$

Доказательство. Докажем, например, первое неравенство:

$$\overline{T}_A(s) = \sum_{x\in X_s} C_A^T(x)\mathsf{P}_s(x) \leqslant \sum_{x\in X_s} (\max_{x\in X_s} C_A^T(x))\mathsf{P}_s(x) =$$

$$= \sum_{x\in X_s} T_A(s)\mathsf{P}_s(x) = T_A(s) \sum_{x\in X_s} \mathsf{P}_s(x) = T_A(s).$$

Второе неравенство доказывается аналогично. □

Ниже в этом параграфе мы рассмотрим временную сложность в среднем ряда алгоритмов.

Пример 5.1. Как было уже установлено (пример 4.2), бинарный алгоритм возведения в степень n требует $\lambda(n) + \lambda^*(n) - 2$ умножений; если в качестве размера взять $m = \lambda(n)$, то сложность в худшем случае будет равна $2m - 2$. Подсчитаем сложность в среднем, привлекая m как размер входа. Всего имеется 2^{m-1} неотрицательных целых битовой длины m; если считать все эти числа равновероятными, то искомая сложность есть

$$\frac{1}{2^{m-1}} \sum_{n=2^{m-1}}^{2^m-1} (\lambda(n) + \lambda^*(n) - 2) = m - 2 + \frac{1}{2^{m-1}} \sum_{n=2^{m-1}}^{2^m-1} \lambda^*(n).$$

Первая цифра каждого из чисел битовой длины m равна 1; количество чисел, имеющих кроме старшей цифры еще k, $0 \leqslant k \leqslant m-1$, ненулевых двоичных цифр, равно $\binom{m-1}{k}$ (т. е. числу сочетаний из $m-1$ по k). Поэтому искомую сложность можно переписать в виде

$$m - 2 + \frac{1}{2^{m-1}} \left(2^{m-1} + \sum_{k=1}^{m-1} k\binom{m-1}{k}\right) = m - 1 + \frac{1}{2^{m-1}} \sum_{k=1}^{m-1} k\binom{m-1}{k}.$$

Легко проверяется, что при $1 \leqslant k \leqslant m-1$ выполнено

$$k\binom{m-1}{k} = (m-1)\binom{m-2}{k-1},$$

и возможно дальнейшее упрощение выражения для сложности:

$$m-1+\frac{m-1}{2^{m-1}}\sum_{k=1}^{m-1}\binom{m-2}{k-1}=m-1+\frac{m-1}{2^{m-1}}\sum_{l=0}^{m-2}\binom{m-2}{l}=\frac{3}{2}(m-1).$$

Если рассматривать $m=\lambda(n)$ как размер входа бинарного алгоритма вычисления a^n, $n\in\mathbb{N}^+$, то сложность в среднем по числу умножений для этого алгоритма есть $\frac{3}{2}(m-1)$.

Таким образом, при больших m сложность в среднем бинарного алгоритма возведения в степень приблизительно равна трем четвертым сложности в худшем случае.

Анализ сложности в среднем для широкого ряда арифметических алгоритмов опирается на асимптотический закон распределения простых чисел, который мы приведем без доказательства.

Асимптотический закон распределения простых чисел. *Для функции $\pi(n)$, значение которой равно количеству простых чисел, меньших данного натурального n, справедливо асимптотическое равенство*

$$\pi(n)\sim\frac{n}{\ln n}. \tag{5.3}$$

Как следствие, вероятность того, что при выборе «наугад» одного из целых чисел $1, 2, \ldots, n$ попадется простое число, асимптотически равна $\frac{1}{\ln n}$.[1]

Пример 5.2. Вновь вернемся к алгоритму пробных делений, предназначенному для распознавания простоты данного $n\geqslant 2$ (примеры 1.2, 4.1). Будем рассматривать в качестве размера входа битовую

[1] Предположение о справедливости этого закона распределения простых чисел высказывалось, в частности, Гауссом и Лежандром. Впоследствии в 1850 г. Чебышевым было доказано, что для некоторых констант c и C

$$c\frac{n}{\ln n}<\pi(n)<C\frac{n}{\ln n}$$

для всех достаточно больших n, и, более того, им было установлено, что можно положить $C=1{,}10555$, $c=0{,}92129$. Асимптотическое равенство (5.3) было полностью доказано в 1896 г. независимо Ж. Адамаром и Ш. Валле Пуссеном. «После доказательства неравенств Чебышева в 1850 г. речь шла, казалось бы, только о более точном нахождении и сближении постоянных c и C. Однако асимптотический закон распределения простых чисел был доказан лишь полвека спустя, в самом конце XIX века, на основании совершенно новых идей, предложенных Риманом...» [39]. В [39, гл. V] приводится элементарное доказательство неравенств Чебышева для $C=6$ и $c=\frac{1}{3}$. Полное доказательство асимптотического закона имеется, например, в [16].

длину m данного числа. На первый взгляд сложность в среднем этого алгоритма могла бы оказаться существенно меньшей, чем сложность в худшем случае, так как для многих входов затраты совсем невелики. Например, для четных входов достаточно одного деления и т. д. Для сложности в худшем случае по числу делений нами была установлена экспоненциальная оценка $\Theta(2^{m/2})$. Существует ли $d \in \mathbb{N}$ такое, что сложность в среднем алгоритма пробных делений как функция от m допускает оценку $O(m^d)$? Мы видим, что для довольно обширного множества входов размера m алгоритм пробных делений работает быстро, и предположение, что для сложности в среднем существует такое число d, выглядит правдоподобным. На самом деле все обстоит не так, потому что среди всех натуральных чисел доля простых (для которых алгоритм пробных делений работает долго) достаточно велика. Оценка количества $V(m)$ простых среди чисел

$$2^{m-1} \leqslant n < 2^m$$

может быть получена из приведенной выше теоремы о распределении простых чисел: воспользовавшись тем, что

$$\pi(2^m) \sim \frac{2^m}{(\ln 2)m}, \quad m \to \infty,$$

а также равенством $V(m) = \pi(2^m) - \pi(2^{m-1})$, получим

$$V(m) \sim \frac{1}{2 \ln 2} \frac{m-2}{m(m-1)} 2^m \sim \frac{1}{(2 \ln 2)m} 2^m. \tag{5.4}$$

Теперь уже экспоненциальность роста сложности в среднем алгоритма пробных делений выводится легко. Обозначим через $D(n)$ количество делений, требующееся алгоритму пробных делений применительно к натуральному числу n. Для любого простого $p \geqslant 2^{m-1}$, $m \geqslant 7$, выполнено $D(p) > 2^{m/2-1}$ (в самом деле, $D(p) = \lfloor \sqrt{p} \rfloor - 1 \geqslant \lfloor 2^{(m-1)/2} \rfloor - 1 > 2^{(m-1)/2} - 2$, и при $m \geqslant 7$ выполнено $2^{(m-1)/2} - 2 \geqslant 2^{m/2-1}$). Интересующая нас сложность в среднем равна $\dfrac{1}{2^{m-1}} \displaystyle\sum_{n=2^{m-1}}^{2^m-1} D(n)$. При $m \geqslant 7$ мы имеем

$$\frac{1}{2^{m-1}} \sum_{n=2^{m-1}}^{2^m-1} D(n) \geqslant \frac{1}{2^{m-1}} \sum_{\substack{2^{m-1} \leqslant p < 2^m \\ p \text{ — простое}}} D(p) \geqslant V(m) 2^{-m/2}. \tag{5.5}$$

Учитывая (5.4), находим, что последняя величина при $m \to \infty$ асимптотически равна

$$\frac{1}{(2 \ln 2)m} 2^{m/2}.$$

Это говорит о том, что исследуемая сложность по числу делений в среднем есть $\Omega\left(\frac{1}{m}2^{m/2}\right)$ и как функция от m эта сложность не может быть оценена как $O(m^d)$ ни при каком $d \in \mathbb{N}$.

Определение 5.2. Вещественная неотрицательная функция $f(m)$, определенная для целых положительных значений аргумента, называется *полиномиально ограниченной*, если существует полином $P(m)$ с вещественными коэффициентами такой, что $f(m) \leqslant P(m)$ для всех $m \in \mathbb{N}^+$. (Очевидно, что мы получим эквивалентное определение, если дополнительно потребуем, чтобы все коэффициенты полинома $P(m)$ были неотрицательными; другие эквивалентные варианты определения: $f(m) = O(m^d)$ для некоторого $d \in \mathbb{N}$ и $f(m) = m^{O(1)}$.)

Доказанное нами можно сформулировать так:

Пусть в качестве размера входа алгоритма пробных делений, предназначенного для распознавания простоты натурального числа n, привлекается $m = \lambda(n)$. Тогда временные сложности этого алгоритма как в худшем случае, так и в среднем по числу делений не являются полиномиально ограниченными функциями от m.

Краткое обсуждение идеи такого алгоритма распознавания простоты числа, который имеет полиномиально ограниченную сложность в худшем случае (тем самым, разумеется, и в среднем), будет проведено в примере 22.2. При этом речь пойдет не об алгебраической сложности по числу делений, мультипликативной сложности и т. д., а о битовой сложности, что приводит к существенно более сильным утверждениям. До этого в гл. 5 будет детально рассмотрено само понятие битовой сложности.

§ 6. Сортировка и конечные вероятностные пространства. Применение формулы полного математического ожидания

При рассмотрении сложности в среднем мы, прежде всего, должны превратить каждое из множеств входов, обладающих фиксированным размером, в вероятностное пространство. Проще работать с конечными вероятностными пространствами, но как обходиться такими пространствами, имея дело, например, с алгоритмами сортировки, входом каждого из которых, при фиксированном размере n, может быть любой массив $x_1, x_2, \ldots, x_n$ с попарно различными элементами? Общий принцип заключается в том, что мы разбиваем все имеющие фиксированный размер n входы на некоторые классы, включая в один

класс входы, неразличимые для данного алгоритма в том смысле, что алгоритм проделывает одну и ту же последовательность операций для любого входа, принадлежащего классу (этот принцип может применяться не только при рассмотрении той или иной сортировки). Число таких классов может оказаться конечным, даже если само множество входов размера n бесконечно. Если это не противоречит другим предположениям, то такие классы можно рассматривать как элементарные события, приписывая каждому из них некоторую вероятность.

На выполнение любой сортировки с помощью сравнений сами значения элементов $x_1, x_2, \ldots, x_n$ не оказывают никакого влияния, важен лишь их относительный порядок. Поэтому в один класс естественно объединять все массивы $x_1, x_2, \ldots, x_n$, имеющие один и тот же порядок элементов по отношению друг к другу. Такой класс может быть представлен перестановкой чисел $1, 2, \ldots, n$ с соответствующим порядком элементов (перестановкой длины n). В дальнейшем для любого $n \in \mathbb{N}^*$ мы будем рассматривать функцию ψ_n, аргументами которой являются любые попарно различные числа $x_1, x_2, \ldots, x_n$, а значением — перестановка $a = (a_1, a_2, \ldots, a_n)$ чисел $1, 2, \ldots, n$, отражающая относительный порядок чисел $x_1, x_2, \ldots, x_n$. Например,

$$\psi_3\left(-\frac{3}{4}, -10, 17\right) = (2, 1, 3). \tag{6.1}$$

Множество всех перестановок чисел $1, 2, \ldots, n$ будет обозначаться через Π_n, так же будет обозначаться и вероятностное пространство, получаемое приписыванием вероятности $\frac{1}{n!}$ каждой из этих перестановок.

Рассмотрим некоторые события в вероятностном пространстве Π_n; вероятность каждого из них, очевидно, будет равна $\frac{M}{n!}$, где M — число перестановок, составляющих событие. Эту вероятность мы будем обозначать через $\mathsf{P}_n(\cdot)$.

Пример 6.1. Начнем с события B_n^ν, $1 \leqslant \nu \leqslant n$, состоящего в том, что ν-й элемент a_ν перестановки $(a_1, a_2, \ldots, a_n)$ равен ν. Перестановку $(a_1, a_2, \ldots, a_n) \in \Pi_n$ можно рассматривать как отображение множества $\{1, 2, \ldots, n\}$ на себя, при котором 1 переходит в a_1, 2 переходит в a_2 и т. д., и событие B_n^ν состоит тогда в том, что ν является неподвижной точкой перестановки. Множество, состоящее из перестановок $(a_1, a_2, \ldots, a_n)$ таких, что $a_\nu = \nu$, содержит $(n-1)!$ элементов, отсюда $\mathsf{P}_n(B_n^\nu) = \frac{(n-1)!}{n!} = \frac{1}{n}$ при всех рассматриваемых ν.

В зависимости от выбранной перестановки число неподвижных точек может быть любым целым из диапазона от 0 до n. Рассмотрим

вопрос о среднем числе неподвижных точек перестановок длины n: найдем математическое ожидание определенной на Π_n случайной величины ξ_n, равной числу неподвижных точек перестановки a. Дополнительно определим случайные величины

$$\zeta_n^\nu = \begin{cases} 1, & \text{если } a_\nu = \nu, \\ 0, & \text{если } a_\nu \neq \nu, \end{cases}$$

$\nu = 1, 2, \ldots, n$. Мы видим, что $\mathsf{P}_n(\zeta_n^\nu = 1) = \mathsf{P}_n(B_n^\nu) = \frac{1}{n}$, откуда $\mathsf{E}\zeta_n^\nu = \frac{1}{n}$ и

$$\mathsf{E}\xi_n = \mathsf{E}(\zeta_n^1 + \zeta_n^2 + \ldots + \zeta_n^n) = \mathsf{E}\zeta_n^1 + \mathsf{E}\zeta_n^2 + \ldots + \mathsf{E}\zeta_n^n = n \cdot \frac{1}{n} = 1.$$

Итак, среднее число неподвижных точек равно 1.[1]

Неподвижные точки перестановки соответствуют элементам массива, которые при сортировке не нуждаются в перемещениях на другие места, они и так находятся именно там, где должны находиться, и это может использоваться в некоторых видах сортировки (см. задачу 25).

Найдем теперь вероятности еще трех событий — эти вероятности потребуются нам при исследовании сложностей в среднем некоторых сортировок.

Предложение 6.1. *Пусть $n \in \mathbb{N}^+$. Тогда*

(i) *при $1 \leqslant u \leqslant n$, $1 \leqslant \nu \leqslant n$ имеем*

$$\mathsf{P}_n(F_n^{u,\nu}) = \frac{1}{n}$$

для события $F_n^{u,\nu}$, состоящего в том, что ν-й элемент перестановки $(a_1, a_2, \ldots, a_n)$ равен u;

(ii) *при $1 < u \leqslant n$ и любой перестановке p чисел $1, 2, \ldots, u-1$ имеем*

$$\mathsf{P}_n(G_n^{u,p}) = \frac{1}{(u-1)!}$$

для события $G_n^{u,p}$, состоящего в том, что относительный порядок элементов $a_{i_1}, a_{i_2}, \ldots, a_{i_{u-1}}$ перестановки $(a_1, a_2, \ldots, a_n)$, меньших u, совпадает с относительным порядком элементов заданной перестановки p (аналогичное утверждение имеет место для элементов, больших u);

(iii) *при $0 \leqslant u < \nu \leqslant n$ имеем*

$$\mathsf{P}_n(H_n^{u,\nu}) = \frac{1}{\nu}$$

[1] Серия вероятностных «задач о совпадении» с решениями приведена в [27].

для события $H_n^{u,v}$, состоящего в том, что в перестановке $(a_1, a_2, \ldots, a_n)$ содержится и элементов a_i, $1 \leqslant i < v$, для которых $a_i < a_v$.

Доказательство. (i) Множество перестановок $(a_1, a_2, \ldots, a_n)$ таких, что $a_v = u$, содержит $(n-1)!$ элементов, независимо от значений u и v.

(ii) Событие $G_n^{u,p}$ включает $\binom{n}{u-1}(n-u+1)!$ перестановок, т. е. $\dfrac{n!}{(u-1)!}$ перестановок, независимо от вида p.

(iii) Очевидно, что если p — любая перестановка чисел $1, 2, \ldots, v$, то множество перестановок $(a_1, a_2 \ldots, a_n)$ таких, что $\psi_v(a_1, a_2, \ldots, a_v) = p$, содержит $\binom{n}{v}(n-v)! = \dfrac{n!}{v!}$ элементов (мы используем функцию ψ_v, определенную перед формулой (6.1)). Заметим, что количество тех перестановок p чисел $1, 2, \ldots, v$, в которых последний элемент равен $u+1$, есть $(v-1)!$. Поэтому количество перестановок, образующих событие $H_n^{u,v}$, равно $\dfrac{n!}{v!}(v-1)!$, т. е. $\dfrac{n!}{v}$, откуда следует требуемое. □

Напомним полезную формулу из курса теории вероятностей. Как и прежде, мы ограничимся рассмотрением конечных вероятностных пространств элементарных событий. Пусть W — такое пространство, P — соответствующее распределение вероятностей, ξ — некоторая случайная величина (функция) на W. Пусть события $W_1, W_2, \ldots, W_l$ задают *полную группу* событий, т. е. эти события попарно несовместны и

$$W = W_1 + W_2 + \ldots + W_l. \tag{6.2}$$

Представление W в виде (6.2) с помощью некоторой полной группы событий мы будем называть *разложением* W. Как обычно, $\mathsf{E}\xi$ — это значение $\sum\limits_{w \in W} \xi(w)\mathsf{P}(w)$, т. е. математическое ожидание (или среднее) случайной величины ξ, а $\mathsf{E}(\xi|W_k)$, $k = 1, 2, \ldots, l$, суть соответствующие *условные математические ожидания* этой случайной величины (при условии осуществления события W_k). Тогда, как известно, имеет место формула *полного математического ожидания*:

$$\mathsf{E}\xi = \sum_{k=1}^{l} \mathsf{E}(\xi|W_k)\mathsf{P}(W_k). \tag{6.3}$$

Оценивание математических ожиданий является оцениванием числовых сумм. В этом иногда помогает простой прием, описанный в приложении В.

Пример 6.2. Исследуем сложности в среднем по числу сравнений и перемещений сортировки простыми вставками, которую, как и прежде, мы будем рассматривать в двух вариантах I_1 и I_2 (см. пример 1.4). Вначале займемся первым вариантом I_1 этой сортировки, полагая затраты равными числу сравнений. Входные массивы считаем перестановками чисел $1, 2, \ldots, n$. Для исследования сложности по числу сравнений введем случайные величины $\xi_n^1, \xi_n^2, \ldots, \xi_n^{n-1}$, определенные так, что значение ξ_n^i для

$$a = (a_1, a_2, \ldots, a_n) \in \Pi_n$$

равно затратам на i-м шаге нашей сортировки, применяемой к a. Ясно, что интересующее нас математическое ожидание есть $\mathsf{E}(\xi_n^1 + \xi_n^2 + \ldots + \xi_n^{n-1})$, и

$$\overline{T}_{I_1}(n) = \mathsf{E}\xi_n^1 + \mathsf{E}\xi_n^2 + \ldots + \mathsf{E}\xi_n^{n-1}. \tag{6.4}$$

Найдем ξ_n^i, $1 \leqslant i < n$. Рассматривая события $H_n^{k,i+1}$, $k = 0, 1, \ldots, i$, образующие разложение пространства Π_n, и применяя формулу (6.3) полного математического ожидания, мы имеем согласно предложению 6.1(iii)

$$\mathsf{E}\xi_n^i = \frac{1}{i+1} \sum_{k=0}^{i} \mathsf{E}(\xi_n^i | H_n^{k,i+1}). \tag{6.5}$$

Если перестановка $a = (a_1, a_2, \ldots, a_n)$ принадлежит событию $H_n^{k,i+1}$, то в этой перестановке перед элементом a_{i+1} имеется ровно k элементов, меньших его, и, очевидно,

$$\xi_n^i = \begin{cases} i-k+1, & \text{если } k > 0, \\ i, & \text{если } k = 0. \end{cases}$$

Мы видим, что при выполнении события $H_n^{k,i+1}$ значение ξ_n^i определено однозначно (напомним, что значение ξ_n^i, $i = 1, 2, \ldots, n-1$, равно числу сравнений на i-м шаге, а перед выполнением i-го шага элементы $a_1, a_2, \ldots, a_i$ уже упорядочены по возрастанию). Поэтому

$$\mathsf{E}(\xi_n^i | H_n^{k,i+1}) = \begin{cases} i-k+1, & \text{если } k > 0, \\ i, & \text{если } k = 0. \end{cases}$$

В соответствии с (6.5)

$$\mathsf{E}\xi_n^i = \frac{1}{i+1}\left(i + \sum_{k=1}^{i}(i-k+1)\right) = \frac{i}{2} + 1 - \frac{1}{i+1}.$$

Используя (6.4), получаем выражение

$$n-1+\sum_{i=1}^{n-1}\Big(\frac{i}{2}-\frac{1}{i+1}\Big)$$

для искомой сложности в среднем. С помощью известной из математического анализа формулы (ее вывод имеется в приложении В)

$$\sum_{i=1}^{n}\frac{1}{i}=\ln n+O(1)$$

сложность $\overline{T}_{I_1}(n)$ можно записать в виде

$$\overline{T}_{I_1}(n)=\frac{(n+4)(n-1)}{4}-\ln n+O(1) \tag{6.6}$$

и, проще, в виде

$$\overline{T}_{I_1}(n)=\frac{n^2}{4}+O(n). \tag{6.7}$$

Таким образом, сложность в среднем по числу сравнений первого варианта сортировки простыми вставками, как и сложность в худшем случае, есть величина порядка n^2, но при больших n первая сложность примерно вдвое меньше второй. Формула (6.6) показывает, что, вообще говоря, вопреки бытовым представлениям сложность в среднем не есть среднее арифметическое затрат в худшем и в лучшем случае: такое среднее арифметическое для рассматриваемой сортировки является рациональной функцией от n и не может описываться этой формулой, содержащей логарифм.

Если рассмотреть вместо случайных величин $\xi_n^1, \xi_n^2, \ldots, \xi_n^{n-1}$ случайные величины $\eta_n^1, \eta_n^2, \ldots, \eta_n^{n-1}$, соответствующим образом отражающие затраты исследуемой сортировки по числу перемещений, то, очевидно, будем иметь

$$\xi_n^i \leqslant \eta_n^i \leqslant \xi_n^i+1,$$

$i=1, 2, \ldots, n-1$. Поэтому сложность в среднем по числу перемещений тоже есть $\frac{n^2}{4}+O(n)$. Нетрудно показать, что и сложности в среднем по числу сравнений и числу перемещений второго варианта сортировки простыми вставками равны $\frac{n^2}{4}+O(n)$. Сложность в среднем по суммарному числу сравнений и перемещений и для первого, и для второго варианта есть $\frac{n^2}{2}+O(n)$, поскольку для любых случайных величин ξ, η, определенных на каком-либо вероятностном пространстве, выполняется

$$\mathsf{E}(\xi+\eta)=\mathsf{E}\xi+\mathsf{E}\eta. \tag{6.8}$$

Каждая из сложностей в среднем как по числу сравнений, так и по числу перемещений для двух вариантов сортировки простыми вставками (*всего четыре сложности*) *есть* $\frac{n^2}{4} + O(n)$; *каждая из сложностей в среднем по общему числу сравнений и перемещений для двух вариантов этой сортировки* (*всего две сложности*) *есть* $\frac{n^2}{2} + O(n)$.

Можно вспомнить, что в примере 1.4, в котором мы рассматривали для двух вариантов сортировки простыми вставками их сложности в худшем случае по суммарному числу сравнений и перемещений, наблюдалась непохожая ситуация, чему имеется, повторимся, простое объяснение: в отличие от (6.8), равенство $\max(\xi + \eta) = \max \xi + \max \eta$, вообще говоря, места не имеет.

Общая формулировка установленного нами соотношения:

Теорема 6.1. *Сложность в среднем некоторого алгоритма по суммарному числу операций нескольких различных типов равна сумме сложностей в среднем этого алгоритма по числу операций каждого типа в отдельности.*

Среди наиболее элементарных сортировок мы пока рассмотрели сложность в среднем только для сортировки простыми вставками. Сразу же можно добавить, что для сортировки выбором ее сложности по числу сравнений в среднем и в худшем случае совпадают, так как число сравнений однозначно определяется длиной n массива. Число сравнений на i-м этапе (i-м просмотре массива) пузырьковой сортировки равно $n - i$. Не сталкиваясь с большими трудностями, можно показать (см. задачу 38), что математическое ожидание $s(n)$ числа просмотров массива есть

$$n - \sum_{k=0}^{n} \frac{k!\, k^{n-k}}{n!}. \tag{6.9}$$

Очевидно, что $s(n) < n$. С другой стороны,

$$\sum_{k=0}^{n} \frac{k!\, k^{n-k}}{n!} \leqslant \sum_{k=0}^{n} \frac{(n-1)!\, k}{n!} = \frac{1}{n} \sum_{k=0}^{n} k = \frac{n+1}{2}.$$

Поэтому $s(n) \geqslant n - \frac{n+1}{2} = \frac{n-1}{2}$. Мы имеем

$$\frac{n-1}{2} \leqslant s(n) < n$$

и $s(n) = \Theta(n)$. Из этого следует, что сложность в среднем по числу сравнений пузырьковой сортировки квадратична.

Сложности в среднем по числу сравнений пузырьковой сортировки, сортировок выбором и простыми вставками имеют одинаковый порядок со сложностями этих же сортировок в худшем случае (допускают оценку $\Theta(n^2)$*).*

§ 7. Пример медленного роста сложности в среднем в сравнении со сложностью в худшем случае

Следующий пример показывает, что при $n \to \infty$ сложность в среднем может расти существенно медленнее, чем сложность в худшем случае.

Пример 7.1. Рассмотрим сложность в среднем $\overline{T}_{\mathrm{QS}}(n)$ быстрой сортировки по числу сравнений. Будем считать, что в качестве разбивающего элемента всегда выбирается первый элемент сортируемой части массива. Введем случайную величину ξ_n на Π_n, равную числу сравнений быстрой сортировки для $a \in \Pi_n$ (в этом смысле $\xi_n(a) = C^T_{\mathrm{QS}}(a)$).

Имеем

$$\overline{T}_{\mathrm{QS}}(n) = \mathsf{E}\xi_n = \frac{1}{n!} \sum_{a \in \Pi_n} \xi_n(a).$$

Введем разложение

$$\Pi_n = K_n^1 + K_n^2 + \ldots + K_n^n,$$

где событию K_n^i принадлежат все те перестановки длины n, для которых разбивающий элемент занимает i-ю позицию после завершения разбиения. В силу предложения 6.1(i) (видно, что $K_n^i = F_n^{i,1}$ — событие состоит в том, что 1-й элемент перестановки равен i), получаем

$$\mathsf{P}_n(K_n^i) = \frac{1}{n}, \quad i = 1, 2, \ldots, n.$$

Разбиение перестановки a (первый этап быстрой сортировки) порождает две перестановки $a' \in \Pi_{i-1}$ и $a'' \in \Pi_{n-i}$. В соответствии с этим определим еще две случайные величины

$$\xi'_n(a) = \xi_{i-1}(a'), \quad \xi''_n(a) = \xi_{n-i}(a''),$$

где i — номер позиции, которую занимает разбивающий элемент после разбиения. По формуле полного математического ожидания

$$\mathsf{E}\xi_n = \sum_{i=1}^{n} \mathsf{E}(\xi_n | K_n^i)\mathsf{P}_n(K_n^i) = \sum_{i=1}^{n} (n - 1 + \mathsf{E}(\xi'_n | K_n^i) + \mathsf{E}(\xi''_n | K_n^i))\frac{1}{n};$$

после очевидного упрощения имеем

$$\mathsf{E}\xi_n = n - 1 + \frac{1}{n}\sum_{i=1}^{n}(\mathsf{E}(\xi'_n|K_n^i) + \mathsf{E}(\xi''_n|K_n^i)).$$

Займемся вычислением $\mathsf{E}(\xi'_n|K_n^i)$ и $\mathsf{E}(\xi''_n|K_n^i)$, $1 \leqslant i \leqslant n$.

Мы имеем разложение

$$K_n^i = \sum_{p\in\Pi_{i-1}} G_n^{i,p},$$

где событие $G_n^{i,p}$ определено так же, как в предложении 6.1(ii), — это событие состоит в том, что относительный порядок элементов перестановки, имеющих значения, меньшие i, совпадает с порядком элементов перестановки p длины $i-1$; имеем

$$\mathsf{E}(\xi'_n|K_n^i) = \sum_{p\in\Pi_{i-1}} \mathsf{E}(\xi'_n|G_n^{i,p})\mathsf{P}_n(G_n^{i,p}).$$

Когда выполняется $G_n^{i,p}$, значение $\xi'_n(a)$ равно $\xi_{i-1}(p)$. Принимая дополнительно во внимание предложение 6.1(ii), имеем

$$\mathsf{E}(\xi'_n|K_n^i) = \sum_{p\in\Pi_{i-1}} \xi_{i-1}(p)\frac{1}{(i-1)!} = \mathsf{E}\xi_{i-1}.$$

Аналогично можно показать, что

$$\mathsf{E}(\xi''_n|K_n^i) = \sum_{p\in\Pi_{n-i}} \xi_{n-i}(p)\frac{1}{(n-i)!} = \mathsf{E}\xi_{n-i};$$

здесь надо рассмотреть событие, которое состоит в том, что относительный порядок элементов перестановки, имеющих значения, большие i, совпадает с порядком элементов перестановки p длины $n-i$.

В итоге получаем

$$\mathsf{E}\xi_n = n - 1 + \frac{1}{n}\sum_{i=1}^{n}(\mathsf{E}\xi_{i-1} + \mathsf{E}\xi_{n-i}).$$

Так как $\sum\limits_{i=1}^{n}\mathsf{E}\xi_{i-1} = \sum\limits_{i=1}^{n}\mathsf{E}\xi_{n-i} = \sum\limits_{k=0}^{n-1}\mathsf{E}\xi_k$, то

$$\mathsf{E}\xi_n = n - 1 + \frac{2}{n}\sum_{k=0}^{n-1}\mathsf{E}\xi_k. \tag{7.1}$$

Таким образом, для $\overline{T}_{\mathrm{QS}}(n) = \mathsf{E}\xi_n$ мы имеем соотношение

$$\overline{T}_{\mathrm{QS}}(n) = n - 1 + \frac{2}{n}\sum_{k=0}^{n-1} \overline{T}_{\mathrm{QS}}(k), \tag{7.2}$$

$\overline{T}_{\mathrm{QS}}(0) = 0$. Домножение обеих частей равенства (7.2) на n дает

$$n\overline{T}_{\mathrm{QS}}(n) = n(n-1) + 2\sum_{k=0}^{n-1} \overline{T}_{\mathrm{QS}}(k).$$

При $n > 0$ имеем для $n-1$:

$$(n-1)\overline{T}_{\mathrm{QS}}(n-1) = (n-1)(n-2) + 2\sum_{k=0}^{n-2} \overline{T}_{\mathrm{QS}}(k).$$

Вычитая почленно последнее равенство из предпоследнего, получаем

$$n\overline{T}_{\mathrm{QS}}(n) - (n-1)\overline{T}_{\mathrm{QS}}(n-1) = 2(n-1) + 2\overline{T}_{\mathrm{QS}}(n-1),$$

т. е.

$$n\overline{T}_{\mathrm{QS}}(n) - (n+1)\overline{T}_{\mathrm{QS}}(n-1) = 2(n-1).$$

Делим последнее равенство на $n(n+1)$ и переходим к новой неизвестной функции $t(n) = \dfrac{\overline{T}_{\mathrm{QS}}(n)}{n+1}$:

$$t(n) - t(n-1) = 2\frac{n-1}{n(n+1)}, \quad t(0) = 0.$$

Решением любого уравнения вида $t(n) - t(n-1) = \varphi(n)$ при функции φ, определенной для всех $n \geqslant n_0$, является $t(n) = \varphi(n_0) + \sum\limits_{k=n_0+1}^{n} \varphi(k)$, $n \geqslant n_0$ (легко доказывается индукцией по n в предположении, что если нижний предел суммирования больше верхнего, то сумма равна нулю). В нашем случае

$$t(n) = 2\sum_{k=1}^{n} \frac{k-1}{k(k+1)} = 2\sum_{k=1}^{n} \frac{1}{k+1} - 2\sum_{k=1}^{n} \frac{1}{k(k+1)}. \tag{7.3}$$

Мы имеем $\sum\limits_{k=1}^{n} \dfrac{1}{k+1} = \ln n + O(1)$; с другой стороны, $\sum\limits_{k=1}^{n} \dfrac{1}{k(k+1)} = O(1)$, поскольку ряд $\sum\limits_{k=1}^{\infty} \dfrac{1}{k(k+1)}$ сходится. Таким образом, $t(n) = 2\ln n + O(1)$; отсюда

$$\overline{T}_{\mathrm{QS}}(n) = (n+1)t(n) = 2n\ln n + O(n). \tag{7.4}$$

В то же время, сложность в худшем случае $T_{\mathrm{QS}}(n)$ есть величина порядка n^2 (см. задачу 10).

Число перемещений, требуемое быстрой сортировкой для конкретного массива, не превосходит числа сравнений, домноженного на некоторую небольшую константу, откуда сложность в среднем $\overline{T}'_{\mathrm{QS}}(n)$ по числу перемещений элементов допускает оценку $\overline{T}'_{\mathrm{QS}}(n) = O(n \log n)$.

Сложность в среднем быстрой сортировки как по числу сравнений, так и по числу перемещений элементов допускает оценку $O(n \log n)$.

Быстрая сортировка не использует дополнительных массивов, поэтому пространственная сложность по числу одновременно хранимых дополнительных величин, равных каким-то элементам исходного массива, ограничена (небольшой) константой и, как следствие, допускает оценку $O(1)$.

Нелишне заметить, что если быстрая сортировка реализована как рекурсивная процедура, то при ее выполнении будет использован стек отложенных заданий. Рекурсию в данном случае можно организовать так, что в стек будут попадать лишь значения индексов некоторых элементов, но не сами элементы массива. В соответствии с определением алгебраической сложности объем такого стека не влияет на пространственную сложность.

Если выйти за рамки алгебраической сложности, то размер стека отложенных заданий становится важной характеристикой алгоритма. Мы рассмотрим размер этого стека для некоторого типа рекурсивных сортировок, к которому относится не только быстрая сортировка, но и, например, рекурсивный вариант сортировки слияниями. Пусть сортировка организована так, что на первом ее этапе мы, следуя некоторому принципу, выделяем из массива x длины $n > 1$ две какие-то его непересекающиеся части x', x'', длина каждой из которых меньше n. На втором этапе эти части сортируются рекурсивно с помощью той же сортировки, и на третьем, заключительном, этапе, исходя из результатов сортировки x' и x'', массив x представляется, уже без рекурсий, в упорядоченном виде. Если $n = 0$ или $n = 1$, то никаких действий над элементами массива не производится.

Будем использовать для такого рода сортировок рабочее название «бинарные рекурсивные сортировки». Возникающие при выполнении таких сортировок рекурсии реализуются с помощью стека отложенных заданий.

Теорема 7.1. *Если некоторая бинарная рекурсивная сортировка такова, что первым из* x', x'' *сортируется массив, имеющий не пре-*

восходящую $\frac{n}{2}$ *длину, то в ходе применения сортировки к массиву* x *длины* $n > 1$ *число отложенных заданий, хранящихся в стеке, никогда не превосходит* $\log_2 n$.

Доказательство. Проведем индукцию по n. При $n = 2$ утверждение, очевидно, справедливо. Пусть утверждение доказано для $2, 3, \ldots, n-1$, и пусть массив x, содержащий n элементов, на первом этапе сортировки порождает два массива x', x'', причем длина n' первого массива x' не превосходит $\frac{n}{2}$, длина n'' второго массива x'' не превосходит $n-1$. Пусть $n'' \geqslant n' \geqslant 1$. На протяжении сортировки массива x', помимо отложенных заданий, связанных с сортировкой x', в стеке будет храниться еще одно задание — сортировать x''. Но в момент, когда сортировка массива x' завершена, в стеке не останется никаких ее следов; таким образом, по предположению индукции, число отложенных заданий в стеке никогда не превзойдет $\max\{\log_2 n' + 1, \log_2 n''\}$. В силу того, что $n' \leqslant \frac{n}{2}$ и $n'' \leqslant n-1$, мы имеем

$$\max\{\log_2 n' + 1, \log_2 n''\} \leqslant \log_2 n,$$

что и требовалось. □

Полученная оценка числа отложенных заданий, хранящихся в стеке, является оценкой для сложности в худшем случае и, тем самым, для сложности в среднем тоже.

Принимая во внимание теорему 7.1, мы приходим к варианту быстрой сортировки, представляемому рекурсивной процедурой $qsort(k, l)$, обращение к которой обеспечивает сортировку части $x_k, x_{k+1}, \ldots, x_l$ исходного массива $x_1, x_2, \ldots, x_n$; при $k \geqslant l$ никаких действий не производится. Сам массив $x_1, x_2, \ldots, x_n$ является глобальным по отношению к процедуре:

```
if k < l then
   x_k выбирается в качестве разбивающего элемента и выполняется разбиение x_k, x_{k+1}, ..., x_l; пусть i — индекс, получаемый разбивающим элементом после разбиения;
   if i − k < l − i then qsort(k, i − 1); qsort(i + 1, l) else
   qsort(i + 1, l); qsort(k, i − 1)
   fi
fi.
```

Быстрая сортировка всего массива является результатом выполнения $qsort(1, n)$.

§ 8. Функция затрат рандомизированного алгоритма

Определение 8.1. Алгоритмы с элементами случайности, реализуемыми обращениями к генераторам случайных чисел, называются *рандомизированными*.

Рандомизированные алгоритмы можно разделить на вероятностные, или, что то же самое, алгоритмы типа *Монте-Карло*, и алгоритмы типа *Лас-Вегас*. Первый тип допускает, что ответ, который дает алгоритм для поставленной задачи с некоторым конкретным входом, может быть неправильным, хотя бы и с малой вероятностью; второй тип — Лас-Вегас — гарантирует правильный ответ, но (как и для алгоритмов типа Монте-Карло) время получения ответа для конкретного входа не определяется, вообще говоря, однозначно этим входом [1]. Исключая специально оговариваемые редкие случаи, мы будем рассматривать только алгоритмы типа Лас-Вегас, не упоминая этого каждый раз.

Анализ сложности рандомизированного алгоритма сводится к нахождению математических ожиданий некоторых случайных величин. Но ситуация здесь отличается от той, когда множество входов фиксированного размера рассматривается как вероятностное пространство и затраты алгоритма, однозначно определенные для каждого конкретного входа, становятся случайными величинами на этом пространстве (мы шли этим путем в двух предыдущих параграфах). При исследовании рандомизированных алгоритмов вероятностное пространство, на котором рассматриваются случайные величины, состоит из *сценариев* выполнения алгоритма для фиксированного входа, и каждый сценарий определяется тем, какие случайные числа будут сгенерированы в соответствующие моменты выполнения алгоритма; за каждым таким сценарием закрепляется некоторая вероятность. В нашем контексте генератор случайных чисел можно представлять себе как стандартную функцию $random(N)$ целого положительного аргумента N, результатом выполнения которой является элемент множества $\{0, 1, \ldots, N-1\}$, но невозможно предсказать точно, каким именно будет значение этой функции, — любой из элементов указанного множества может появиться с вероятностью $1/N$.

Таким образом, затраты рандомизированного алгоритма при фиксированном входе, вообще говоря, не определяются однозначно, но

[1] Эта классификация является достаточно распространенной в специальной литературе по рандомизированным алгоритмам — см., например, книгу [55], — но не единственной. Например, в [26, разд. 9.2] рандомизированные алгоритмы подразделяются иначе, а именно — на алгоритмы типа Монте-Карло, типа Лас-Вегас и шервудские. Мы не будем останавливаться на этом.

зависят от сценария вычисления. При фиксированном входе мы можем рассмотреть множество всех сценариев и, приписав адекватным образом каждому из сценариев некоторую вероятность, ввести на полученном вероятностном пространстве случайную величину, значение которой для данного сценария равно соответствующим вычислительным затратам. Значение функции затрат на данном входе можно положить равным математическому ожиданию этой случайной величины (*усредненным* затратам для данного входа). А после того как определена функция затрат и принято соглашение о том, что такое размер входа, мы можем, как обычно, рассматривать сложность алгоритма — в худшем случае или же в среднем (последнее возможно, если только множество входов каждого фиксированного размера является вероятностным пространством).

Мы не будем здесь сколь-либо глубоко входить в общие проблемы теории сложности рандомизированных алгоритмов, а ограничимся рассмотрением примеров, в которых вероятностное пространство сценариев является одним и тем же для каждого из входов данного размера (в общем случае это не так).

Пример 8.1. Вновь обратимся к быстрой сортировке. В § 7 нами установлено, что быстрая сортировка имеет сравнительно низкую сложность в среднем в предположении, что для каждого $n > 0$ все относительные порядки элементов входных массивов длины n имеют одинаковую вероятность $\frac{1}{n!}$. Но в некоторых практических задачах это предположение может оказаться безосновательным в силу того, что все входные массивы изначально оказываются, например, «почти упорядоченными». Число сравнений, затрачиваемых быстрой сортировкой на каждом таком «почти упорядоченном» массиве, будет близко к n^2, что сведет на нет достоинства быстрой сортировки.

Однако если разбивающий элемент выбирать случайно, то при каждом фиксированном входе размера n усредненные затраты будут иметь порядок $n \log n$, что и будет показано ниже. Итак, под рандомизированной быстрой сортировкой мы будем понимать быструю сортировку со случайным выбором индекса разбивающего элемента; если надо выбрать случайное значение m из $k, k+1, \ldots, l$, то мы полагаем $m := k + random(l - k + 1)$. В том алгоритме, который записан в виде процедуры в конце § 7, вместо

> x_k выбирается в качестве разбивающего элемента и выполняется разбиение $x_k, x_{k+1}, \ldots, x_l$; пусть i — индекс, получаемый разбивающим элементом после разбиения;

можно написать

> $m := k + random(l - k + 1)$;
> x_m выбирается в качестве разбивающего элемента и выполняется разбиение $x_k, x_{k+1}, \ldots, x_l$; пусть i — индекс, получаемый разбивающим элементом после разбиения;

это даст рандомизированный вариант быстрой сортировки.

Пусть дан массив $x_1, x_2, \ldots, x_n$ попарно различных чисел. Пусть мы выбираем элемент m из множества $\{1, 2, \ldots, n\}$ и вероятность выбора каждого из элементов есть $\frac{1}{n}$. Тогда место x_m в массиве $x_1, x_2, \ldots, x_n$ после сортировки этого массива может оказаться любым с одной и той же вероятностью $\frac{1}{n}$. В самом деле, пусть $1 \leqslant i \leqslant n$ и пусть $l(i)$ таково, что $x_{l(i)}$ занимает после сортировки массива место с номером i. Это $l(i)$ определяется единственным образом. Поэтому вероятность попадания x_m после сортировки массива на i-е место равна вероятности того, что $m = l(i)$. Последняя вероятность, очевидно, равна $\frac{1}{n}$. Следовательно, приписывание одинаковой вероятности каждому возможному выбору индекса разбивающего элемента ведет к тому, что вероятность каждого из событий «после этапа разбиения элемент, выступавший в качестве разбивающего, занимает в массиве i-е место», $i = 1, 2, \ldots, n$, равна $\frac{1}{n}$.

Это обстоятельство позволяет переформулировать задачу анализа сложности рандомизированной быстрой сортировки, например, следующим образом. Имеется полоска бумаги шириной в одну клетку и длиной в n клеток, $n \geqslant 0$, которую надо разрезать на отдельные клетки. Разрезы имеет право производить некий разрезальщик, которому надо платить за работу. При $n = 0$ или $n = 1$ разрезальщик ничего не делает и ничего не получает. В случае $n > 1$ он выбирает случайным образом (тянет из шапки билет с номером) значение i из множества $\{1, 2, \ldots, n\}$ и затем вырезает из полоски i-ю клетку (рис. 8), получая за этот этап работы вознаграждение в $n - 1$ рубль. Кроме вырезанной клетки возникают две полоски, длина каждой из

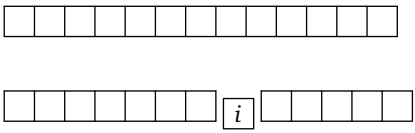


Рис. 8. Этап разрезания полоски.

которых не превышает $n-1$, при этом не исключается равенство нулю одной из длин. С каждой из двух полосок разрезальщик проделывает ту же операцию, получая вознаграждение в соответствии с тем же принципом оплаты. Каково математическое ожидание размера вознаграждения, получаемого разрезальщиком за всю работу?

Здесь при фиксированном n множество всех возможных сценариев (будем называть их n-сценариями) — это фактически множество

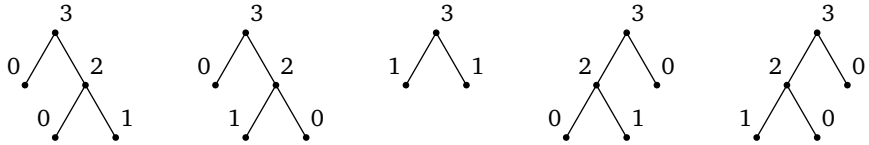


Рис. 9. Сценарии разрезания полоски из трех клеток (3-сценарии).

некоторых двоичных деревьев; на рис. 9 показано одно из возможных представлений множества всех 3-сценариев. В каждой вершине дерева записывается число клеток в полоске. При произвольном $n \geqslant 0$ понятие n-сценария определяется рекурсивно: 0-сценарий и 1-сценарий — это одна вершина, которой приписано число 0 или соответственно 1; пусть $n > 1$, тогда любой n-сценарий s получается выбором некоторого числа i, $1 \leqslant i \leqslant n$, некоторого $(i-1)$-сценария s' и некоторого $(n-i)$-сценария s'': из корня n исходят ребра к вершинам $i-1$ и $n-i$, к которым подклеиваются корни сценариев s' и s'' (рис. 10). Каждому n-сценарию s естественным образом приписывается вероятность $\mathsf{P}_n(s)$ (здесь индекс n указывает на то, что вероятность соотнесена с некоторым n-сценарием): если $n=0$ или $n=1$, то сценарии имеют вероятность 1, а при $n>1$

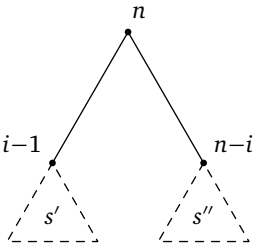


Рис. 10. Структура n-сценария.

$$\mathsf{P}_n(s) = \frac{1}{n}\mathsf{P}_{i-1}(s')\mathsf{P}_{n-i}(s''). \tag{8.1}$$

Например, третий из 3-сценариев, изображенных на рис. 9, имеет вероятность $\frac{1}{3}$, каждый из остальных — вероятность $\frac{1}{6}$.

Формула (8.1) отражает идею алгоритма, а именно что после определения i выбор $(i-1)$-сценария s' и выбор $(n-i)$-сценария s'' независимы друг от друга. Эта формула превращает множество всех n-сценариев при фиксированном n в вероятностное пространство S_n. Поль-

зуясь индукцией по n, проверим, например, что $\sum\limits_{s\in S_n} \mathsf{P}_n(s) = 1$: при $n=0$ и $n=1$ это очевидно, при $n>1$ имеем

$$\sum_{s\in S_n} \mathsf{P}_n(s) = \sum_{i=1}^{n} \sum_{\substack{s'\in S_{i-1}\\ s''\in S_{n-i}}} \frac{1}{n}\mathsf{P}_{i-1}(s')\mathsf{P}_{n-i}(s'') =$$

$$= \frac{1}{n}\sum_{i=1}^{n}\Big(\Big(\sum_{s'\in S_{i-1}} \mathsf{P}_{i-1}(s')\Big)\Big(\sum_{s''\in S_{n-i}} \mathsf{P}_{n-i}(s'')\Big)\Big) = \frac{1}{n}\sum_{i=1}^{n} 1\cdot 1 = 1.$$

Плата за разрезание полоски длины n на отдельные клетки полностью определяется использованным n-сценарием, это задает случайную величину χ_n на S_n, и нас интересует математическое ожидание этой величины. Для сценария s, представленного на рис. 10, мы назовем s' и s'' левым и правым *подсценариями*. При $n>1$ определим на S_n дополнительно к χ_n еще две случайные величины χ'_n и χ''_n, положив $\chi'_n(s) = \chi_{i-1}(s')$ и $\chi''_n(s) = \chi_{n-i}(s'')$, где s' и s'' суть левый и правый подсценарии сценария s. Имеем при $n>1$

$$\chi_n(s) = n - 1 + \chi'_n(s) + \chi''_n(s).$$

Пусть $n>1$. Введем разложение

$$S_n = S_n^1 + S_n^2 + \ldots + S_n^n, \tag{8.2}$$

где S_n^i — это событие «левый подсценарий данного n-сценария является $(i-1)$-сценарием (и соответственно правый подсценарий является $(n-i)$-сценарием)», $i = 1, 2, \ldots, n$. Очевидно,

$$\mathsf{P}_n(S_n^1) = \mathsf{P}_n(S_n^2) = \ldots = \mathsf{P}_n(S_n^n) = \frac{1}{n},$$

поэтому по формуле полного математического ожидания получаем

$$\mathsf{E}\chi_n = \sum_{i=1}^{n} \mathsf{E}(\chi_n|S_n^i)\frac{1}{n} =$$

$$= n - 1 + \frac{1}{n}\sum_{i=1}^{n}(\mathsf{E}(\chi'_n|S_n^i) + \mathsf{E}(\chi''_n|S_n^i)) =$$

$$= n - 1 + \frac{1}{n}\sum_{i=1}^{n}(\mathsf{E}\chi_{i-1} + \mathsf{E}\chi_{n-i}) =$$

$$= n - 1 + \frac{2}{n}\sum_{i=1}^{n}\mathsf{E}\chi_{i-1} = n - 1 + \frac{2}{n}\sum_{k=0}^{n-1}\mathsf{E}\chi_k.$$

Итак,

$$\mathsf{E}\chi_n = n - 1 + \frac{2}{n}\sum_{k=0}^{n-1}\mathsf{E}\chi_k, \quad \mathsf{E}\chi_0 = 0.$$

Такого вида соотношения уже встречались в предыдущем параграфе, и по аналогии с (7.1) мы находим

$$\mathsf{E}\chi_n = 2n\ln n + O(n). \tag{8.3}$$

Возвратимся к рандомизированной быстрой сортировке. Для данного массива длины n математическое ожидание затрат, измеряемых числом сравнений, полностью определяется значением n. Поэтому, например, сложность в худшем случае, т. е. максимум математических ожиданий рассматриваемого вида для массивов длины n, есть просто математическое ожидание, найденное при рассмотрении этого n. Мы можем переписать (8.3) как соотношение для сложности $\widetilde{T}_{\mathrm{QS}}(n)$ по числу сравнений рандомизированной быстрой сортировки:

$$\widetilde{T}_{\mathrm{QS}}(n) = 2n\ln n + O(n).$$

Вновь можно заметить, что число перемещений, требуемое рандомизированной быстрой сортировкой для конкретного массива, не превосходит числа сравнений, домноженного на некоторую константу, откуда сложность в среднем $\widetilde{T}'_{\mathrm{QS}}(n)$ по числу перемещений элементов допускает оценку $\widetilde{T}'_{\mathrm{QS}}(n) = O(n\log n)$. Как и обычная быстрая сортировка, рандомизированная быстрая сортировка не использует дополнительных массивов, поэтому $\tilde{S}_{\mathrm{QS}}(n) = O(1)$.

Сложность рандомизированной быстрой сортировки как по числу сравнений, так и по числу перемещений элементов допускает оценку $O(n\log n)$. Пространственная сложность по числу одновременно хранимых дополнительных величин, равных каким-то элементам исходного массива, допускает оценку $O(1)$.

При использовании принципа «из двух заданий первым выполняется более простое», мы получаем, согласно теореме 7.1, для сложности по объему стековой памяти оценку $\log_2 n$.

Совпадению сложностей $\overline{T}_{\mathrm{QS}}(n)$ и $\widetilde{T}_{\mathrm{QS}}(n)$ можно дать объяснение: случайный выбор разбивающего элемента, предпринимаемый на каждом этапе сортировки, эквивалентен тому, что мы меняем порядок исходного массива случайным образом и затем применяем «обычную» быструю сортировку; но равенство сложностей такого рода не обязано иметь место для других алгоритмов, — все-таки речь идет о математических ожиданиях случайных величин, определенных

на разных вероятностных пространствах. Например, можно несколько ускорить рандомизированную быструю сортировку, если разбивающий элемент определять путем случайного выбора без возвращения трех элементов массива и последующего определения второго по величине из этих трех; можно показать, что тогда вместо $2n \ln n + O(n)$ возникнет $\frac{12}{7} n \ln n + O(n)$ [1], и совпадения сложностей обычной и рандомизированной быстрой сортировки уже не будет.

Возвращаясь к примеру 3.1, мы видим, что, основываясь на рандомизированной быстрой сортировке, можно получить алгоритм построения выпуклой оболочки данных n точек координатной плоскости, имеющий сложность в среднем $O(n \log n)$ по общему числу операций.

При нахождении или оценивании сложности какого-либо рандомизированного алгоритма часто не возникает необходимости детального рассмотрения всевозможных сценариев. Бывает достаточно рассмотреть сходное с (8.2) разложение пространства сценариев и применить формулу полного математического ожидания. Даже если множество всех сценариев бесконечно, разложение вида (8.2) можно выбрать конечным.

Пример 8.2. Массив $x_1, x_2, \ldots, x_n$ назовем массивом, *содержащим большинство*, если больше половины его элементов имеют равные значения. Пусть над элементами массивов разрешена операция проверки равенства двух элементов, будем называть эту операцию сравнением. Пусть для данного массива, содержащего большинство, требуется найти i, $1 \leqslant i \leqslant n$, такое, что значение x_i принадлежит большинству значений элементов $x_1, x_2, \ldots, x_n$. Один из возможных рандомизированных алгоритмов решения этой задачи состоит в том, что i выбирается случайно (распределение вероятностей равномерно), а затем проверяется, удовлетворяет ли x_i поставленному условию, — сравнений на эту проверку уйдет не более $n - 1$. Если x_i не подходит, то все повторяется. Сложность в среднем по числу сравнений этого алгоритма не превосходит $2(n - 1)$. В самом деле, пространство всех сценариев разлагается на два события: первая попытка найти нужное i оказывается удачной и, соответственно, эта попытка оказывается неудачной. По формуле полного математического ожидания имеем для искомого среднего a:

$$a \leqslant p(n-1) + (1-p)(a+n-1).$$

Отсюда $a \leqslant \frac{1}{p}(n-1) < 2(n-1)$.

[1] Доказательство имеется, например, в [59, разд. 3.7].

Еще один путь получения этого же результата. Так как вероятность p удачного выбора индекса i больше половины, то математическое ожидание количества попыток, которые потребуются для получения удачного значения i, меньше двух. Отсюда среднее число сравнений a меньше, чем $2(n-1)$.

Теоретически нет никакого препятствия к тому, чтобы при определении функции затрат рандомизированного алгоритма взять вместо математического ожидания максимум затрат по соответствующим сценариям (при таком подходе рандомизированная быстрая сортировка имеет квадратичную сложность). Но обычно для рандомизированных алгоритмов рассматриваются усредненные затраты.

Добавим в заключение, что правомерен взгляд на рандомизированные алгоритмы, при котором каждому возможному входу сопоставляется вероятностное пространство обычных детерминированных алгоритмов[1]. Если в этой связи вернуться к примеру 8.1, то можно заметить, что для разрезания полоски бумаги каждый из рассматриваемых сценариев задает конкретный детерминированный алгоритм разрезания.

Мы вернемся к этому в § 18.

Задачи

23. Верно ли, что для рассмотрения сложности в среднем некоторого алгоритма требуется задание распределения вероятностей

а) на множестве всех допустимых входов?

б) на каждом из множеств всех входов фиксированного размера?

24. Как говорилось при обсуждении асимптотического закона распределения простых чисел, в [39] приводится элементарное доказательство неравенств Чебышева с $C=6$, $c=1/3$. Добавим, что наиболее легко доказывается нижняя оценка

$$\pi(n) > \frac{1}{3}\frac{n}{\ln n},$$

справедливая для всех $n>1$. Доказать, что сложность в среднем алгоритма пробных делений не является полиномиально ограниченной, используя для этого только последнее неравенство и не прибегая к полному варианту асимптотического закона распределения простых чисел.

[1] См., например, [55].

Указание. Предположение о том, что $V(m) < 2^{3m/4}$ для всех $m > m_0$, позволило бы, исходя из равенства

$$\pi(2^m) = \pi(2^{m_0}) + V(m_0+1) + V(m_0+2) + \ldots + V(m),$$

вывести верхнюю оценку $\pi(2^m) = O(2^{3m/4})$, что вступило бы в противоречие с неравенством $\pi(2^m) \geqslant c\dfrac{2^m}{m}$ с положительной константой c. Поэтому существует последовательность $m_1 < m_2 < \ldots$ натуральных чисел таких, что $V(m_i) \geqslant 2^{3m_i/4}$, $i = 1, 2, \ldots$ Можно рассмотреть (5.5) для $m = m_1, m_2, \ldots$

25. Сложность по числу перемещений (обменов) в среднем для сортировки выбором при условии, что мы исключаем обмены вида $x_k \leftrightarrow x_k$, не превосходит $n - 2 + \dfrac{1}{n}$, где n — длина массива.

Указание. В перестановках из Π_n среднее число элементов a_i, $1 \leqslant i \leqslant n-1$, таких, что $a_i = i$, есть $\dfrac{n-1}{n}$ (см. пример 6.1).

26. Найти среднее число присваиваний $m := \ldots$ при выполнении алгоритма поиска наименьшего элемента массива:

```
m := x_1;
for i = 2 to n do
   if x_i < m then m := x_i fi
od
```

Предполагается, что все возможные взаимные порядки элементов в исходном массиве длины n являются равновероятными; в качестве размера входа берется n.

Указание. Рассмотреть разложение всего вероятностного пространства в сумму двух событий: последний элемент исходного массива является его наименьшим элементом (вероятность равна $\dfrac{1}{n}$) и соответственно последний элемент исходного массива не является его наименьшим элементом (вероятность равна $\dfrac{n-1}{n}$). Пусть $s(n)$ обозначает искомое математическое ожидание. Показать, что условное математическое ожидание исследуемого числа присваиваний, соответствующее первому событию, есть $s(n-1)+1$, а условное математическое ожидание, соответствующее второму событию, есть $s(n-1)$. Пользуясь формулой полного математического ожидания, вывести отсюда рекуррентное соотношение для $s(n)$ и найти его решение, удовлетворяющее условию $s(1) = 1$.

27. Пользуясь дискретным аналогом формулы Ньютона—Лейбница, согласно которому $\sum\limits_{k=1}^{n} (f(k+1) - f(k)) = f(n+1) - f(1)$, вычислить сумму $\sum\limits_{k=1}^{n} \dfrac{1}{k(k+1)}$, входящую в (7.3).

28. Исходя из (7.3), пользуясь приемом оценки сумм из приложения В и результатом из предыдущей задачи, доказать неравенство $\overline{T}_{\mathrm{QS}}(n) \leqslant 2n \ln n$, из которого будет следовать, что величина $O(n)$ в правой части (7.4) отрицательна.

29. Вернемся к задаче 8. В качестве алгоритма с указанными свойствами можно предложить следующий: при наличии вагонов на правой стороне узла к операции В прибегаем лишь тогда, когда на левой стороне невозможно увеличить на 1 число вагонов с правильным чередованием цветов с помощью какой-то из операций МИМО и ИЗ; в остальных случаях выполняется одна из операций МИМО и ИЗ, если возможны обе, то первая из них. Как и в задаче 8, считаем, что черных и белых вагонов по n штук, и n — размер входа. Какова сложность по числу сортировочных операций этого алгоритма в среднем?

Указание.[1] Можно доказать следующие три утверждения.

1) Предположим, что в исходном расположении первый вагон на правой стороне белый. Тогда число сортировочных операций, требующихся алгоритму, равно $3n - k$, где k есть количество максимальных сплошных последовательностей черных вагонов в исходном расположении (непосредственно слева и справа от каждой такой сплошной последовательности нет черных вагонов, таков смысл «максимальности»).

2) Вновь предположим, что в исходном расположении первый вагон на правой стороне белый. Тогда количество исходных расположений, в которых имеется ровно k максимальных сплошных последовательностей черных вагонов, есть $\binom{n}{k}\binom{n-1}{k-1}$.

3) $\sum\limits_{k=1}^{n-1}\binom{n-1}{n-k}\binom{n-1}{k} = \binom{2n-2}{n}$.

Из первых двух утверждений выводится, что искомая сложность в среднем есть

$$2n + 2 \cdot \frac{\sum\limits_{k=1}^{n-1}(n-k)\binom{n}{k}\binom{n-1}{k-1}}{\binom{2n}{n}},$$

третье утверждение позволяет получить из этого, что интересующая нас сложность есть $\frac{n(5n-3)}{2n-1} = \frac{5}{2}n + \frac{1}{4} + o(1)$.

30. (Задача не связана со сложностью в среднем, но приводит к рекуррентному соотношению, которое решается сходно с (7.2).) Доказать, что для любого $l > 0$ можно, не применяя клей, построить из костяшек домино, длина каждой из которых равна 2, навес длины l

[1] Решение принадлежит А. П. Фокину.

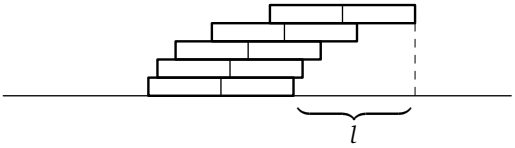


Рис. 11. Навес из костяшек домино.

(рис. 11). Предложить алгоритм конструирования таких навесов, требующий $O(e^l)$ костяшек.

Указание. Будем характеризовать навес из n костяшек неотрицательными числами $l_1, l_2, \ldots, l_n$, где l_i — расстояние от правого края i-й сверху костяшки до вертикальной (пунктирной на рис. 11) линии, проходящей через правый край 1-й, т. е. самой верхней, костяшки. (Таким образом, $l_1 = 0$, $l_n = l$.) Считая, что вес каждой костяшки равен 1, можно показать, что условием равновесия является система неравенств

$$l_{i+1} \leqslant \frac{(l_1+1)+(l_2+1)+\ldots+(l_i+1)}{i}, \quad i = 1, 2, \ldots, n-1$$

(каждое такое неравенство означает, что центр тяжести конструкции из i верхних костяшек не выходит за правый край $(i+1)$-й костяшки). Самый длинный навес получается тогда, когда числа $l_1, l_2, \ldots$ подчинены рекуррентному соотношению

$$l_{i+1} = \frac{(l_1+1)+(l_2+1)\ldots+(l_i+1)}{i}, \quad l_1 = 0.$$

Далее можно идти тем же путем, что и при решении соотношения (7.2) и т. д.

31. Перестановка $(a_1, a_2, \ldots, a_n) \in \Pi_n$ называется *полной*, если $a_i \neq i$, $i = 1, 2, \ldots, n$. Вероятность $d_n = \mathsf{P}_n(D_n)$ события D_n, состоящего в том, что данная перестановка является полной, равна 0 при $n = 1$ и равна

$$\frac{1}{2!} - \frac{1}{3!} + \ldots + \frac{(-1)^n}{n!} \tag{8.4}$$

при $n > 1$ (следствие: $\lim\limits_{n\to\infty} d_n = \frac{1}{e}$).

Задача о значении d_n широко известна как задача о шляпах. На собрание, состоявшееся поздно вечером, n его участников пришли в шляпах и оставили их в раздевалке, а когда собрание окончилось, разобрали шляпы наугад в темноте (в раздевалке перегорела лампочка) и разошлись по домам. Какова вероятность того, что каждый пришел домой в чужой шляпе?

Указание. Для любого r такого, что $0 \leqslant r \leqslant n$, можно рассмотреть вероятность $p(n, r)$ того, что перестановка длины n имеет ровно r непо-

движных точек (ровно r человек пришли домой в своих шляпах). Тогда $p(n,0)+p(n,1)+\ldots+p(n,n)=1$. Далее надо доказать, что $p(n,r)=$ $=\binom{n}{r}\frac{1}{n(n-1)\ldots(n-r)}p(n-r,0)=\frac{1}{r!}p(n-r,0)$, откуда будет следовать, что

$$\frac{1}{0!}d_n+\frac{1}{1!}d_{n-1}+\frac{1}{2!}d_{n-1}+\ldots+\frac{1}{(n-1)!}d_{n-1}+\frac{1}{n!}d_0=0.$$

С учетом $d_0=1$ последнее соотношение однозначно определяет все $d_1, d_2, \ldots$ Остается убедиться, что подстановка значений (8.4) в левую часть этого соотношения дает 0. Подставляя $\frac{1}{0!}-\frac{1}{1!}+\frac{1}{2!}-\frac{1}{3!}+\ldots+\frac{(-1)^r}{r!}$, $r=0,1,\ldots,n$, в левую часть этого соотношения, получим $\sum\limits_{j=0}^{n}\sum\limits_{i=0}^{n-j}\frac{(-1)^i}{j!\,i!}$ (роль r играет $n-j$). Слагаемые этой суммы можно перегруппировать так, чтобы внутри группы значение $i+j$ было постоянной величиной l. Это даст

$$\sum_{l=0}^{n}\sum_{i=0}^{l}\frac{(-1)^i}{(l-i)!\,i!}=\sum_{l=0}^{n}\Big(\frac{1}{l!}\sum_{i=0}^{l}\frac{l!}{(l-i)!\,i!}(-1)^i\Big),$$

последний шаг доказательства очевиден.

32. При быстрой сортировке (в любом из рассмотренных ее вариантов) число пар индексов (i,j), попадающих в стек отложенных заданий, не превосходит длины n исходного массива.

33. Верно ли, что определение усредненных затрат некоторого рандомизированного алгоритма требует задания распределения вероятностей

а) на множестве всех допустимых входов?

б) на каждом из множеств всех входов фиксированного размера?

34. Если измерять затраты рандомизированной быстрой сортировки количеством обращений к генератору случайных чисел, то сложность этой сортировки есть величина порядка n. То же самое верно, если в определении функции затрат вместо математического ожидания взять максимум затрат по соответствующим сценариям.

35. Идея, лежащая в основе быстрой сортировки, может быть использована для нахождения m-го наименьшего элемента среди $a_1, a_2, \ldots, a_n$, $1\leqslant m\leqslant n$ (самый маленький элемент — это 1-й наименьший, следующий элемент — 2-й наименьший и т. д.). Разбиение массива с использованием случайно выбранного разбивающего элемента порождает два массива длины $i-1$ и $n-i$ при некотором $1\leqslant i\leqslant n$. Если $m=i$, то поиск закончен, если $m\leqslant i-1$, то далее рекурсивно находится m-й наименьший в первом массиве, а если $m>i$, то рекурсивно находится $(m-i)$-й наименьший во втором массиве. Можно ввести сложность по числу сравнений при рассмотрении двух параметров

n, m размера входа. Беря максимум этих сложностей по всем m таким, что $1 \leqslant m \leqslant n$, определяем сложности $T(n)$, $\overline{T}(n)$ по числу сравнений в худшем случае и в среднем с использованием n в качестве размера входа. Показать, что $T(n) = \Theta(n^2)$ и $\overline{T}(n) < 4n$.

Указание. Возьмем $U(n) = \max\limits_{1 \leqslant k \leqslant n} \overline{T}(k)$. Очевидно, что $U(n)$ не убывает при возрастании n и что $\overline{T}(n) \leqslant U(n)$. Доказывается неравенство

$$U(n) \leqslant \frac{1}{n}(U(n-1) + \max\{U(1), U(n-2)\} + \ldots \\ \ldots + \max\{U(n-2), U(1)\} + U(n-1)) + n - 1,$$

из которого, в силу неубывания $U(n)$, следует

$$U(n) \leqslant \frac{2}{n}\left(U(n-1) + U(n-2) + \ldots + U\left(\left\lfloor \frac{n}{2} \right\rfloor\right)\right) + n.$$

Отсюда с помощью индукции выводится, что $U(n) < 4n$ для всех $n \geqslant 1$.

36. В *инверсионном векторе* $(b_1, b_2, \ldots, b_n)$ произвольной перестановки $(a_1, a_2, \ldots, a_n) \in \Pi_n$ каждая компонента b_i равна количеству целых j таких, что $1 \leqslant j < i$ и $a_j > a_i$. Например, инверсионный вектор перестановки $(2, 4, 3, 1, 6, 5)$ — это $(0, 0, 1, 3, 0, 1)$. Показать, что каждому целочисленному вектору $(b_1, b_2, \ldots, b_n)$, для которого $0 \leqslant b_i < i$, $i = 1, 2, \ldots, n$, соответствует некоторая перестановка длины n, для которой этот вектор является инверсионным.

Указание. Очевидно, что если $b_n = k$, $1 \leqslant k < n$, то $a_n = n - k$, и это соображение приводит к алгоритму построения $(a_1, a_2, \ldots, a_n)$, применимому к любому вектору b, удовлетворяющему оговоренным условиям: просматриваем компоненты вектора b, продвигаясь от последней к первой, находим значение соответствующей компоненты перестановки a и удаляем найденное значение из множества M, первоначально равного $\{1, 2, \ldots, n\}$; при этом значение a_i, $i = n, n-1, \ldots, 1$, определяется как $(b_i - 1)$-й наибольший в множестве M (самый большой элемент — это первый наибольший, следующий по величине элемент — это второй наибольший и т. д.).

37. Показать, что один этап пузырьковой сортировки понижает на единицу значение каждой неотрицательной компоненты инверсионного вектора перестановки (см. предыдущую задачу) и не изменяет нулевые компоненты. Показать, что вероятность того, что значение максимума компонент инверсионного вектора выбранной наугад перестановки длины n равно k, $0 \leqslant k < n$, есть $\dfrac{k^{n-k}k!}{n!}$.

Указание ко второй части задачи. Количество векторов $(b_1, b_2, \ldots, b_n)$, для каждого из которых $b_i \in \mathbb{N}^+$, $0 \leqslant b_i < i$, $i = 1, 2, \ldots, n$, и $\max\{b_1, b_2, \ldots, b_n\} \leqslant k$,

$1 \leqslant k \leqslant n-1$, равно $k!\,k^{n-k-1}$, так как b_i может принимать любое значение между 0 и $i-1$ при $i \leqslant k$ и любое значение между 0 и $k-1$ для $k < i \leqslant n$.

38. Показать, что математическое ожидание числа этапов пузырьковой сортировки совпадает с (6.9).

Указание. Пользуясь решением предыдущей задачи, легко получить, что это математическое ожидание есть

$$\frac{1}{n!}\sum_{k=1}^{n}(k(k^{n-k}k! - (k-1)^{n-k+1}(k-1)!),$$

что равно

$$\frac{1}{n!}\Bigl(\sum_{k=1}^{n}(k^{n-k+1}k! - (k-1)^{n-k+2}(k-1)!) - \sum_{k=1}^{n}(k-1)^{n-k+1}(k-1)!\Bigr).$$

В упрощении последнего выражения поможет дискретный аналог формулы Ньютона—Лейбница (см. задачу 27).

39. Используя идею решения задачи 35, предложить рандомизированный алгоритм восстановления перестановки длины n по ее инверсионному вектору (см. задачу 36), имеющий сложность в среднем по числу сравнений $O(n^2)$.

40. Пусть $(a_1, a_2, \ldots, a_n)$ — произвольная перестановка длины n. Ее преобразование

$$\texttt{for } i = n \texttt{ downto } 2 \texttt{ do } j := 1 + random(i-1)\texttt{;}\ a_i \leftrightarrow a_j \texttt{ od}$$

может дать любую перестановку длины n с вероятностью $\frac{1}{n!}$. (Это дает алгоритм построения случайных перестановок с равномерным распределением вероятностей.)

41. Предположим, что у нас нет другого генератора случайных чисел, кроме генератора, в результате обращения к которому появляется 0 или 1 с одинаковой вероятностью $\frac{1}{2}$ (аналог подбрасывания монеты), и пусть p, $0 \leqslant p \leqslant 1$, — заданное вещественное число. Каким образом с помощью этого генератора можно определить генератор *randp*, в результате обращения к которому появляется 0 или 1 с вероятностями соответственно p и $1-p$ (незначительные отклонения допустимы)? Сложность в среднем алгоритма получения одного случайного числа с помощью *randp* должна быть меньше 2 (затраты определяются числом обращений к изначально имеющемуся генератору).

Указание. Представить p (возможно, с небольшой погрешностью) в виде конечной суммы вида $\frac{a_1}{2} + \frac{a_2}{2^2} + \ldots + \frac{a_k}{2^k}$, где a_i — это 0 или 1 для всех i, а k — некоторое целое положительное число.

42. Пусть изначально у нас имеется генератор случайных вещественных чисел, принадлежащих отрезку $[0, 1]$, и вероятность появления числа, принадлежащего отрезку $[a, b]$, $0 \leqslant a \leqslant b \leqslant 1$, есть $b - a$. Пусть даны неотрицательные вещественные числа $\alpha_0, \alpha_1, \ldots, \alpha_{N-1}$ такие, что $\alpha_0 + \alpha_1 + \ldots + \alpha_{N-1} = 1$. Как определить генератор чисел, принадлежащих множеству $\{0, 1, \ldots, N-1\}$, такой, что вероятность появления числа k, $0 \leqslant k \leqslant N-1$, равна α_k?

43. Предположим, что у нас нет другого генератора случайных чисел, кроме генератора, в результате обращения к которому появляется 0 с вероятностью p или 1 с вероятностью $1 - p$, причем о p мы ничего не знаем, кроме того, что $p \neq 0$ и $p \neq 1$. Как с помощью этого генератора можно сконструировать генератор, в результате обращения к которому появляется 0 или 1 с одинаковой вероятностью $1/2$?

Указание. Порождая подряд две цифры с помощью имеющегося генератора, мы получаем комбинации $0, 1$ и $1, 0$ с одинаковой ненулевой вероятностью.

44. (Продолжение предыдущей задачи.) Чему равно математическое ожидание числа обращений к изначально имеющемуся генератору случайных чисел при построении последовательности пар до появления $0, 1$ или $1, 0$? Найти сложность в среднем алгоритма получения k «равновероятных» нулей и единиц с помощью сконструированного генератора (затраты определяются количеством обращений к изначально имеющемуся генератору). Можно ли указать значения p, для которых эта сложность имеет минимальное и, соответственно, максимальное значение?

45. Предложенную в указании к задаче 43 идею обобщить на случай, когда необходимо сконструировать генератор $random(N)$, $N \geqslant 1$, описанный в § 8. Найти сложность в среднем алгоритма получения k равновероятных случайных чисел из отрезка $[0, N-1]$ (затраты определяются числом обращений к изначально имеющемуся генератору).

46. Сохранится ли для сложности в среднем формула $2n \ln n + O(n)$, если в задаче о разрезании полоски клетчатой бумаги на отдельные клетки (см. пример 8.1) считать, что плата за вырезание одной случайно выбранной клетки равна не $n-1$, а n рублей? Тот же вопрос, если эта плата составляет n^2 рублей.

47. Известное утверждение (теорема Дирихле, 1849 г.), что два случайных числа $x, y \in \mathbb{N}^+$ с вероятностью P, равной $\frac{6}{\pi^2}$, оказываются взаимно простыми, имеет тот смысл, что если ввести в рассмотрение число $M(n)$, равное количеству пар (x, y) таких, что x и y взаимно

просты и, дополнительно, $1 \leqslant x, y \leqslant n$, то предел $\lim\limits_{n\to\infty} \dfrac{M(n)}{n^2}$ существует и равен $\dfrac{6}{\pi^2}$. Предполагая (не обосновывая и не вникая в то, можно ли это обосновать; это соответствует «физическому уровню строгости»), что множество $\mathbb{N}^+$ может быть превращено в вероятностное пространство так, что случайное $x \in \mathbb{N}^+$ кратно фиксированному $d \in \mathbb{N}^+$ с вероятностью $\dfrac{1}{d}$, можно предложить несколько довольно простых доказательств равенства $P = \dfrac{6}{\pi^2}$. Для этого можно воспользоваться тем, что

$$\sum_{n=1}^{\infty} \frac{1}{n^2} = \frac{\pi^2}{6}$$

(доказано Эйлером в 1734 г.) или свойством дзета-функции Римана, согласно которому

$$\sum_{n=1}^{\infty} \frac{1}{n^s} = \prod_{p\ \text{— простое}} \frac{1}{1-p^s}$$

и которое справедливо, например, для всех целых $s > 1$, и, в частности, для $s = 2$.

В упомянутых выше предположениях доказать равенство $P = \dfrac{6}{\pi^2}$, используя следующие подходы.

а) Для произвольного $d \in \mathbb{N}^+$ вычислить вероятность $\varphi(d)$ того, что для случайных $x, y \in \mathbb{N}^+$ выполнено $d = \text{нод}(x, y)$. Из равенства $P = 1 - \sum\limits_{d=2}^{\infty} \varphi(d)$ определить P.[1]

б) Для произвольного простого p вычислить вероятность $\psi(p)$ того, что по крайней мере одно из случайных $x, y \in \mathbb{N}^+$ не делится на p. Из равенства $P = \prod\limits_{p\ \text{— простое}} \psi(p)$ определить P.[2]

Указания. а) $\varphi(d) = \dfrac{P}{d^2}$; б) $\psi(p) = 1 - \dfrac{1}{p^2}$.

Для того чтобы сделать доказательство «настоящим», надо для произвольного $n \in \mathbb{N}^+$ детально рассмотреть ситуацию $1 \leqslant x, y \leqslant n$ и перейти к пределу при $n \to \infty$, что потребует более кропотливой работы.[3]

[1] См. [19, разд. 4.5.2].
[2] См. [50, гл. I, разд. 3, прим. 3.3].
[3] См. [19, разд. 4.5.2, упр. 10].

Глава 3

Оценивание числа шагов (итераций) алгоритма

§ 9. Функции, убывающие по ходу выполнения алгоритма

Часто выполнение алгоритма является последовательностью однотипных шагов (итераций). Если все шаги равнозатратны по времени, то общее их число с точностью до постоянного множителя эквивалентно временным затратам рассматриваемого алгоритма для данного входа, и важной задачей является определение или хотя бы оценивание числа этих шагов.

Пример 9.1. Пусть a_0, a_1 — натуральные числа, $a_0 \geqslant a_1$. Поиск наибольшего общего делителя (нод) чисел a_0, a_1 по алгоритму Евклида требует выполнения серии однотипных шагов, каждый из которых — деление с остатком:

$$\begin{aligned} a_0 &= q_1 a_1 + a_2, \\ a_1 &= q_2 a_2 + a_3, \\ &\dots\dots\dots\dots\dots \\ a_{n-3} &= q_{n-2} a_{n-2} + a_{n-1}, \\ a_{n-2} &= q_{n-1} a_{n-1} + a_n, \\ a_{n-1} &= q_n a_n. \end{aligned} \tag{9.1}$$

В этом случае $a_n = \text{нод}(a_0, a_1)$, так как

$$\text{нод}(a_0, a_1) = \text{нод}(a_1, a_2) = \dots = \text{нод}(a_n, 0) = a_n.$$

Последовательность получаемых остатков убывает (остаток меньше делителя), при этом все остатки — неотрицательные целые. Не существует убывающей бесконечной последовательности, элементами которой являются неотрицательные целые числа, поэтому выполнение алгоритма Евклида (будем обозначать его буквой Е) завершается для любых натуральных a_0, a_1, и число делений с остатком не пре-

восходит a_1. Обозначив через $C_{\mathrm{E}}(a_0, a_1)$ исследуемое число делений с остатком, получаем

$$C_{\mathrm{E}}(a_0, a_1) \leqslant a_1 \tag{9.2}$$

(для упрощения записи мы пишем $C_{\mathrm{E}}(a_0, a_1)$ вместо $C_{\mathrm{E}}^T(a_0, a_1)$). Это говорит о том, что если a_1 рассматривается как размер входа или если a_0, a_1 рассматриваются как два параметра размера входа, то сложность алгоритма Евклида по числу делений не превосходит a_1. Как мы увидим, эта оценка является весьма грубой, но, привлекая сходные рассуждения, можно получать и более тонкие оценки.

Формализуем те соображения, которые привели нас к этой первой оценке. Каждый шаг алгоритма имеет дело с парой неотрицательных целых (a_{i-1}, a_i) и при условии $a_i \neq 0$ перерабатывает эту пару в (a_i, a_{i+1}), где a_{i+1} равно остатку от деления a_{i-1} на a_i. На множестве пар неотрицательных целых чисел k, l мы определяем функцию $L(k, l)$, принимающую неотрицательные целые значения. Эта функция такова, что когда при выполнении алгоритма Евклида мы переходим от пары (a_{i-1}, a_i) к паре (a_i, a_{i+1}), значения функции убывают: $L(a_{i-1}, a_i) > L(a_i, a_{i+1})$. Так как функция целочисленна, то ее значение убывает с каждым шагом по крайней мере на единицу. Отсюда следует, что общее число шагов не превзойдет значения функции L на исходной паре чисел. Мы рассмотрели функцию $L(k, l) = l$, и это привело нас к выводу, что число шагов не превзойдет значения $L(a_0, a_1) = a_1$.

Для обоснования того, что число делений в алгоритме Евклида растет медленнее, чем a_1, было бы достаточно найти соответствующую функцию $L(k, l)$, значения которой при больших k, l оказываются существенно меньшими, чем l. Эта функция по-прежнему должна быть определена для любых неотрицательных целых k, l, $k \geqslant l$, принимать неотрицательные целые значения и убывать при замене пары k, l парой l, r, где r — остаток от деления k на l. Хотя бы одна пара целых неотрицательных k, l, $k \geqslant l$, должна обращать $L(k, l)$ в нуль, иначе вместо $L(k, l)$ можно взять функцию $L'(k, l) = L(k, l) - 1$ и т. д.

Предложение 9.1. *Пусть $k, l \in \mathbb{N}^+$, $k > l$, и пусть r — остаток от деления k на l. Тогда*

(i) *$\lambda(k) > \lambda(r)$, где, как обычно, $\lambda(\cdot)$ — битовая длина данного числа*;

(ii) *функция*

$$L(k, l) = \lambda(k) + \lambda(l) - 2 \tag{9.3}$$

такова, что $L(k, l) > L(l, r)$.

Доказательство. (i) Имеем $k = ql + r$, $q \geqslant 1$, откуда $k \geqslant l + r > 2r$. Следовательно, $\lambda(k) > \lambda(r)$.

(ii) Из $\lambda(k) > \lambda(r)$ следует $\lambda(k) + \lambda(l) - 2 > \lambda(l) + \lambda(r) - 2$. □

Очевидно, что функция (9.3) неотрицательна. Таким образом, справедлива оценка

$$C_{\mathrm{E}}(a_0, a_1) \leqslant \lambda(a_0) + \lambda(a_1) - 2. \tag{9.4}$$

Но если a_0 очень велико в сравнении с a_1, то $\lambda(a_0) + \lambda(a_1) - 2 > a_1$. Тем не менее, теперь для $C_{\mathrm{E}}(a_0, a_1)$ уже легко выводится хорошая оценка. Пусть число делений с остатком больше 1. Имеем

$$C_{\mathrm{E}}(a_0, a_1) = C_{\mathrm{E}}(a_1, a_2) + 1 \leqslant \lambda(a_1) + \lambda(a_2) - 1 \leqslant 2\lambda(a_1) - 1.$$

Оценка

$$C_{\mathrm{E}}(a_0, a_1) \leqslant 2\lambda(a_1) - 1, \tag{9.5}$$

очевидно, верна и в случае единственного деления: $C_{\mathrm{E}}(a_0, a_1) = 1$ при том, что $\lambda(a_1) \geqslant 1$.

Если взять a_1 за размер входа (a_0, a_1), то для сложности по числу делений будем иметь

$$T_{\mathrm{E}}(a_1) = \max_{a_0 \geqslant a_1} C_{\mathrm{E}}(a_0, a_1) \leqslant 2\lambda(a_1) - 1 = 2\lfloor \log_2 a_1 \rfloor + 2 - 1 \leqslant 2 \log_2 a_1 + 1.$$

Доказанное нами можно переформулировать так:

Если рассматривать a_1 как размер входа a_0, a_1 алгоритма Евклида, то для сложности $T_{\mathrm{E}}(a_1)$ по числу делений выполнено неравенство

$$T_{\mathrm{E}}(a_1) \leqslant 2 \log_2 a_1 + 1. \tag{9.6}$$

Эта же оценка имеет место и при рассмотрении двух параметров a_0, a_1 размера входа.

Для достаточно больших a_1 верхняя оценка (9.6) существенно лучше, чем $T_{\mathrm{E}}(a_1) \leqslant a_1$. (Несколько более точная, чем (9.6), оценка дается в задаче 52.)

Рассматривая далее оценки сложности по числу делений алгоритма Евклида, мы будем использовать значение a_1 в качестве размера входа (значение a_0 может быть очень большим в сравнении с a_1, но первое же деление с остатком приведет к a_1, a_2, где $a_2 < a_1$).

Известно, что алгоритм Евклида допускает разнообразные обобщения и расширения. Прежде всего, вместе с $\mathrm{нод}(a_0, a_1)$ можно находить целые s, t такие, что

$$sa_0 + ta_1 = \mathrm{нод}(a_0, a_1). \tag{9.7}$$

Чтобы это показать, обратимся к (9.1). Пусть уже известны $a_0, a_1, \ldots$ $\ldots, a_i$, $1 \leqslant i \leqslant n-1$, и пусть для них найдены множители $s_0, t_0; s_1, t_1; \ldots$ $\ldots; s_i, t_i$ такие, что

$$s_0a_0+t_0a_1=a_0,\ \ s_1a_0+t_1a_1=a_1,\ \ \ldots,\ \ s_{i-1}a_0+t_{i-1}a_1=a_{i-1},\ \ s_ia_0+t_ia_1=a_i.$$

После выполнения очередного деления с остатком $a_{i-1} = q_i a_i + a_{i+1}$ имеем $a_{i+1} = a_{i-1} - q_i a_i$ и

$$(s_{i-1} - q_i s_i)a_0 + (t_{i-1} - q_i t_i)a_1 = a_{i+1}.$$

Можем положить

$$s_{i+1} = s_{i-1} - q_i s_i, \quad t_{i+1} = t_{i-1} - q_i t_i, \quad i = 1, \ldots, n-1; \tag{9.8}$$

при этом

$$s_0 = 1, \quad t_0 = 0; \quad s_1 = 0, \quad t_1 = 1. \tag{9.9}$$

Решение поставленной задачи дают числа

$$s = s_n, \quad t = t_n.$$

В процессе применения этого алгоритма к числам $a_0 = 39, a_1 = 15$ возникают остатки $9, 6, 3, 0$ и соответствующие ненулевым остаткам пары множителей суть $1, -2; -1, 3; 2, -5$. Имеем $2 \cdot 39 + (-5) \cdot 15 = 3 =$ $= \text{нод}(39, 15)$.

Этот алгоритм мы назовем *расширенным* алгоритмом Евклида и будем его обозначать буквами EE, от его английского названия extended Euclidean — расширенный евклидов. Каждый шаг расширенного алгоритма Евклида содержит три мультипликативных операции — одно деление с остатком и два умножения.

Если рассматривать a_1 как размер входа a_0, a_1 расширенного алгоритма Евклида, то для мультипликативной сложности $T_{\text{EE}}(a_1)$ этого алгоритма имеем $T_{\text{EE}}(a_1) \leqslant 6 \log_2 a_1 + 3$.

Расширенный алгоритм Евклида дает возможность решать в целых числах линейные уравнения с целыми коэффициентами (см. задачу 56), он также играет существенную роль в модулярной арифметике (см. § 22).

Если дополнить расширенный алгоритм Евклида еще одним шагом, то получатся s_{n+1}, t_{n+1} такие, что

$$s_{n+1}a_0 + t_{n+1}a_1 = 0. \tag{9.10}$$

Например, для $a_0 = 39, a_1 = 15$ имеем $s_5 = -5, t_5 = 13$. Легко доказать, что

$$|s_1| \leqslant |s_2| \leqslant |s_3|, \quad |t_1| \leqslant |t_2| < |t_3|, \tag{9.11}$$

и при $n > 2$

$$|s_3| < |s_4| < \ldots < |s_{n+1}|, \quad |t_3| < |t_4| < \ldots < |t_{n+1}|. \tag{9.12}$$

Несколько труднее, но также возможно доказать, что $|s_i|$ и $|t_i|$ взаимно просты при $i = 1, 2, \ldots, n+1$ (см. задачу 57). Из (9.10) и взаимной простоты s_{n+1} и t_{n+1} следует

$$|s_{n+1}| = \frac{a_1}{\text{нод}(a_0, a_1)}, \quad |t_{n+1}| = \frac{a_0}{\text{нод}(a_0, a_1)}.$$

Вместе с (9.11), (9.12) это дает нам

$$|s_n| \leqslant a_1, \quad |t_n| < a_0. \tag{9.13}$$

Эти неравенства пригодятся нам в дальнейшем.

Пример 9.2. Бинарный поиск места (т. е. значения p, $1 \leqslant p \leqslant n+1$, — как объяснено в приложении А) элемента y в упорядоченном массиве из n элементов $x_1 < x_2 < \ldots < x_n$:

$p := 1$; $q := n+1$;
`while` $p < q$ `do`
  $r := \left\lfloor \frac{p+q}{2} \right\rfloor$; `if` $x_r < y$ `then` $p := r+1$ `else` $q := r$ `fi`
`od`

Обозначим этот алгоритм буквами BS от его английского названия binary search — бинарный поиск. Будем считать число элементов сегмента массива *длиной* этого сегмента (при рассмотрении задач сортировки и поиска мы называем *сегментом* массива любую упорядоченную часть $x_s < x_{s+1} < \ldots < x_{t-1} < x_t$ данного массива). Легко видеть, что от сегмента длины k мы переходим к сегменту длины $\left\lfloor \frac{k}{2} \right\rfloor$ или $\left\lfloor \frac{k}{2} \right\rfloor - 1$. Это говорит о том, что с каждым шагом алгоритма функция $L(k) = \lambda(k)$, где положительное k является длиной сегмента, рассматриваемого на данном шаге, убывает по крайней мере на единицу, пока не приходим к сегменту, содержащему один элемент, после чего еще одно сравнение полностью решает задачу. Отсюда сложность бинарного поиска не превосходит $\lambda(n) = \lfloor \log_2 n \rfloor + 1$. Мы видим также, что если y меньше минимального элемента рассматриваемого сегмента длины $k > 1$, то длина сегмента на следующем шаге будет равна $\left\lfloor \frac{k}{2} \right\rfloor$. Поэтому если изначально $y < x_1$, то в ходе бинарного поиска будут рассматриваться сегменты длины

$$n, \left\lfloor \frac{n}{2} \right\rfloor, \left\lfloor \frac{\left\lfloor \frac{n}{2} \right\rfloor}{2} \right\rfloor, \ldots, 1$$

соответственно (битовая длина каждого следующего элемента последовательности на единицу меньше битовой длины предыдущего), и в целом потребуется в точности $\lambda(n) = \lfloor \log_2 n \rfloor + 1$ сравнений. Используя утверждение, содержащееся в задаче 2, мы можем сформулировать установленное свойство бинарного поиска так:

Сложность $T_{\mathrm{BS}}(n)$ бинарного поиска места элемента в массиве длины n по числу сравнений равна $\lfloor \log_2 n \rfloor + 1$, или, что то же самое, $\lceil \log_2 (n+1) \rceil$.

Выражение $\lceil \log_2 (n+1) \rceil$ для указанной сложности иногда бывает удобнее, чем $\lfloor \log_2 n \rfloor + 1$, потому что оно имеет смысл и в вырожденном случае $n = 0$.

Бинарный поиск находит широчайшее применение при поиске информации в разнообразных таблицах. Укажем здесь еще одно его применение, касающееся вычислительной геометрии: он позволяет быстро определять, принадлежит ли произвольная точка Q выпуклому n-угольнику, заданному вершинами $P_1, P_2, \ldots P_n$. Можно легко построить какую-нибудь внутреннюю точку O данного n-угольника. В силу его выпуклости точка Q — внутренняя, если и только если Q и O лежат в одной полуплоскости относительно любой из прямых $P_1P_2, \ldots, P_{n-1}P_n, P_nP_1$. Это соображение приводит к имеющему временную сложность $\Theta(n)$ алгоритму. Но допустим, что проведены n добавочных лучей (рис. 12), исходящих из точки O и проходящих через

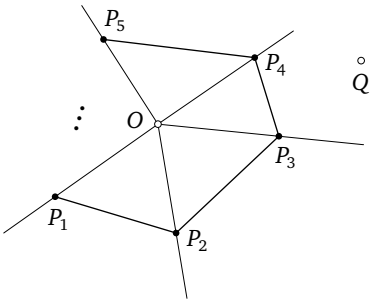


Рис. 12. Точка Q лежит между двумя лучами, проведенными из внутренней точки O многоугольника через его вершины.

вершины (считаем, что $O \neq Q$, иначе мы сразу бы заключили, что Q принадлежит многоугольнику). Можно установить, какому из углов $\angle P_1OP_2, \ldots, \angle P_{n-1}OP_n, \angle P_nOP_1$ принадлежит точка Q: если углы прону-

мерованы в указанном порядке, то бинарным поиском определяется номер m угла, $1 \leqslant m \leqslant n$; при этом если Q лежит на одном из проведенных лучей, то из двух значений m берется любое. Узнав m, мы проверяем согласованность расположения точек O и Q по отношению к прямой, являющейся продолжением той стороны многоугольника, концы которой лежат на сторонах m-го угла.

Теперь заметим, что в самом проведении лучей $OP_1, OP_2, \ldots, OP_n$ нет необходимости: сравнение $\angle P_1OQ$ с $\angle P_1OP_i$ требует ограниченного числа операций и в том случае, когда нам лишь известны координаты точек O, Q, P_1, P_i.

Основывающийся на бинарном поиске алгоритм распознавания принадлежности точки выпуклому n-угольнику имеет сложность $O(\log n)$ по общему числу операций и пространственную сложность $O(1)$.

Полезным для решения ряда задач является то обстоятельство, что если точка не принадлежит данному выпуклому n-угольнику, то с помощью этого алгоритма мы дополнительно определяем одну из сторон n-угольника, которая из этой точки видна целиком (рис. 13).

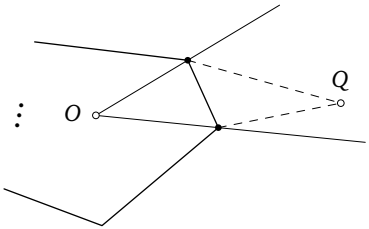


Рис. 13. Для точки Q, не принадлежащей данному выпуклому n-угольнику, находим сторону n-угольника, которая из Q видна целиком.

Пример 9.3. Установим число этапов слияния при сортировке, предложенной Дж. фон Нейманом (которая является одним из вариантов сортировки слияниями). При сортировке фон Неймана шаг за шагом происходят перебросы элементов исходного массива в дополнительный массив и наоборот, и каждая переброска сопровождается слиянием соседних сегментов массива (рис. 14). В данном случае в качестве вспомогательного размера массива удобно рассмотреть k — число сегментов (первоначально $k = n$, затем, шаг за шагом, k довольно быстро убывает). При анализе бинарного поиска места элемента мы фактически использовали, что если $2^{m-1} \leqslant k < 2^m$,

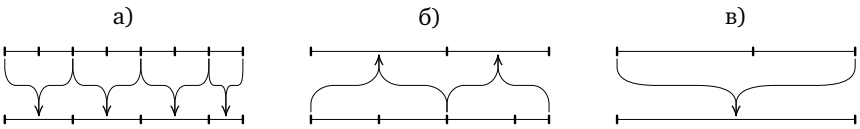


Рис. 14. Три последовательных шага сортировки фон Неймана, удлиняющих упорядоченные участки (сегменты): а) переход от семи сегментов к четырем; б) переход от четырех сегментов к двум; в) переход от двух сегментов к одному (массив полностью упорядочен).

то при $k > 1$ на следующем шаге мы будем иметь дело с k' таким, что $k' < 2^{m-1}$. В случае же сортировки фон Неймана ситуация такова, что если на каком-то шаге имеем $k > 1$ уже построенных сегментов и $2^{m-1} < k \leqslant 2^m$ (это соответствует тому, что $\lceil \log_2 k \rceil = m$), то на следующем шаге сегментов будет $k' < k$ и $2^{m-2} < k' \leqslant 2^{m-1}$ (это соответствует тому, что $\lceil \log_2 k' \rceil = m - 1$). Поэтому функция $L(k) = \lceil \log_2 k \rceil$, где k — текущее число построенных сегментов, убывает с каждым шагом ровно на единицу. Таким образом, число этапов слияния, или число переброс сортируемых элементов из массива в массив, равно $\lceil \log_2 n \rceil$. Это позволяет утверждать следующее:

Сложность сортировки фон Неймана по числу сравнений меньше, чем $n\lceil \log_2 n \rceil$*; сложность по числу присваиваний равна* $n\lceil \log_2 n \rceil$.

В самом деле, каждый этап слияния, сопровождаемый переброской элементов из массива в массив, требует n присваиваний. Число сравнений при каждом слиянии по крайней мере на единицу меньше, чем длина получаемого упорядоченного сегмента: известно, что число тех сравнений, которые априори могут потребоваться для слияния двух упорядоченных массивов, содержащих соответственно k и m элементов, $k \leqslant m$, лежит в диапазоне от k до $k + m - 1$ (обе указанные границы достижимы).

Вновь обращаясь к примеру 3.1, мы видим, что, основываясь на сортировке фон Неймана, можно получить алгоритм построения выпуклой оболочки данных n точек координатной плоскости, имеющий сложность $O(n \log n)$ в худшем случае по общему числу операций.

Некоторое различие между примером 9.1 и примерами 9.2, 9.3 состоит в том, что в примере 9.1 мы определяли L как функцию самого входа алгоритма и из полученной оценки для затрат выводили оценку для сложности (переходили от входа как такового к его размеру), а в 9.2, 9.3 мы изначально рассматривали L как функцию размера.

В этих двух последних примерах значения функции L возникали из сравнений размера n с элементами некоторой последовательности s_k, $k = 0, 1, \ldots$ (в обоих случаях было $s_k = 2^k$). В примере 9.2 удобно было считать, что $L(n) = k$, если $s_{k-1} \leqslant n < s_k$, а в примере 9.3 — если $s_{k-1} < n \leqslant s_k$.

Типичный итерационный алгоритм содержит цикл, в котором набор v начальных величин преобразуется в набор v', затем v' преобразуется в v'' и т. д. Функция L, которую мы подбираем, должна принимать неотрицательные значения и быть определенной либо на самих наборах величин, либо на их размерах; в обоих случаях функция должна убывать по ходу выполнения алгоритма:

$$L(v) > L(v') > L(v'') > \ldots$$

в первом случае, или

$$L(\|v\|) > L(\|v'\|) > L(\|v''\|) > \ldots$$

во втором. Значение $L(v)$ или соответственно $L(\|v\|)$ дает верхнюю границу для числа шагов цикла. [1]

Заметим попутно, что, исходя из установленной в примере 9.2 сложности бинарного поиска, мы сразу можем получить верхнюю оценку

$$T_{\mathrm{B}}(n) \leqslant (n-1)\lceil \log_2 n \rceil \tag{9.14}$$

для сложности по числу сравнений сортировки бинарными вставками; мы используем для обозначения этой сортировки букву B, от английского слова binary — бинарный. По числу перемещений сложность этой сортировки квадратична, здесь эта сортировка уступает сортировке фон Неймана. Пространственную сложность сортировки бинарными вставками можно считать равной нулю (все делается на том же месте), а для сортировки фон Неймана эта сложность есть n.

Пример 9.4. Число итераций численных алгоритмов тоже часто оценивают сравнением размеров данных, соответствующих текущим итерациям, с элементами последовательности, которая может иметь, например, вид q^n или q^{2^n} $(q < 1)$, $n = 0, 1, \ldots$, и т. д. Этим способом

[1] Такую функцию можно назвать *функцией Ляпунова* цикла, подразумевая аналогию с функцией, убывающей вдоль решения обыкновенного дифференциального уравнения (аналогия принадлежит С. С. Лаврову, см. [24, разд. 3.1.3]). Это не единственная аналогия между дифференциальными уравнениями и циклическими алгоритмами и программами. Например, понятие первого интеграла, или закона сохранения, сопоставимо с понятием инварианта цикла и т. д.

получаются оценки числа итераций для классических методов (алгоритмов) приближенного решения уравнений. Вспомним две такие оценки.

Пусть корень уравнения $f(x)=0$ разыскивается на отрезке $[a,b]$, на котором функция $f(x)$ непрерывна, $f(a)f(b)\leqslant 0$. Ошибка не должна превышать данного $\varepsilon>0$. При $\varepsilon\to 0$ и фиксированных $f(x), a, b$ число итераций метода делений пополам есть

$$\log_2 \frac{1}{\varepsilon}+O(1). \tag{9.15}$$

Метод же Ньютона (касательных) требует

$$O\left(\log\log\frac{1}{\varepsilon}\right) \tag{9.16}$$

итераций при соблюдении ограничений на функцию $f(x)$, обеспечивающих сходимость метода.

§ 10. Качество оценок

После вывода оценки сложности алгоритма часто возникает вопрос о том, насколько эта оценка хороша и нельзя ли входящие в оценку константы и функции заменить другими так, чтобы оценка стала более точной, и, как следствие, несущей больше информации об исследуемом алгоритме.

Пример 10.1. В примере 9.1 для сложности $T_{\mathrm{E}}(a_1)$ по числу делений алгоритма Евклида мы легко получили верхнюю оценку a_1, а затем, после некоторых усилий, пришли к логарифмической верхней оценке (9.6). Отвлекаясь от значений констант, перейдем от (9.6) к

$$T_{\mathrm{E}}(a_1)=O(\log a_1). \tag{10.1}$$

Нами пока не доказано, что оценка (10.1) является точной, и, как следствие, что не верна, скажем, оценка $O(\sqrt{\log a_1})$ или $O(\log\log a_1)$ и т. д. Чтобы доказать точность (10.1), достаточно подобрать для алгоритма Евклида последовательность входов

$$(a_0^{(1)},a_1^{(1)}),\quad (a_0^{(2)},a_1^{(2)}),\quad \ldots$$

таких, что последовательность $a_1^{(n)}$ (последовательность размеров входа) неограниченно возрастает и $C_{\mathrm{E}}(a_0^{(n)},a_1^{(n)})=\Omega(\log a_1^{(n)})$ при $n\to\infty$; тогда, очевидно, будем иметь $T_{\mathrm{E}}(a_1^{(n)})=\Omega(\log a_1^{(n)})$.

Фактически, нам нужна последовательность таких входов возрастающего размера, для каждого из которых алгоритм Евклида работает медленно. Подойдет, например, последовательность входов

(F_{n+1}, F_n), $n = 0, 1, \ldots$, где $F_0, F_1, F_2, \ldots$ — числа Фибоначчи, определяемые равенствами $F_0 = 0$, $F_1 = 1$ и правилом «каждое следующее число равно сумме двух предыдущих», т. е. рекуррентным соотношением $F_{n+2} = F_{n+1} + F_n$. Из определения чисел Фибоначчи следует, что применение алгоритма Евклида к F_{n+1}, F_n при $n \geqslant 1$ требует n делений (все частные будут равны единице), поэтому $C_{\mathrm{E}}(F_{n+1}, F_n) = n$.

По индукции доказывается, что для всех неотрицательных n справедлива формула Бине:

$$F_n = \frac{1}{\sqrt{5}}(\phi^n - (-\phi)^{-n}), \tag{10.2}$$

где

$$\phi = \frac{1+\sqrt{5}}{2} = 1{,}61803\ldots$$

(деление отрезка в отношении $1 : \phi$ называют *золотым сечением*). Так как $\phi > 1$, то из (10.2) получаем

$$\phi^n = \sqrt{5}F_n(1 + o(1)), \tag{10.3}$$

и, поскольку $\log_\phi(1 + o(1)) = o(1)$, с помощью логарифмирования (10.3) по основанию ϕ имеем

$$C_{\mathrm{E}}(F_{n+1}, F_n) = n = \log_\phi F_n + O(1) = \Omega(\log F_n),$$

откуда следует, что оценка (10.1) точная.

Мы докажем более сильное утверждение:

$$T_{\mathrm{E}}(a_1) > \frac{1}{4}\log_\phi a_1 + \gamma, \tag{10.4}$$

где γ — некоторая константа. Это вместе с (10.1) даст

$$T_{\mathrm{E}}(a_1) = \Theta(\log a_1). \tag{10.5}$$

Приступая к доказательству, мы перейдем от функции $C_{\mathrm{E}}(a_0, a_1)$, имеющей два аргумента, к функции одного аргумента. Каждому входу (a_0, a_1), $a_0 \geqslant a_1 > 0$, мы сопоставим рациональное число $r = \frac{a_0}{a_1} \geqslant 1$, которое однозначно определяет число шагов алгоритма Евклида для a_0, a_1. (Пусть $\frac{a_0}{a_1} = \frac{\tilde{a}_0}{\tilde{a}_1}$, $\tilde{a}_0$ и $\tilde{a}_1$ взаимно просты, $a_0 = u\tilde{a}_0$, $a_1 = u\tilde{a}_1$; тогда остатки $a_2, a_3, \ldots$ и $\tilde{a}_2, \tilde{a}_3, \ldots$, возникающие в ходе применения алгоритма Евклида к a_0, a_1 и соответственно к $\tilde{a}_0, \tilde{a}_1$, отличаются лишь множителем u.) Будем обозначать это число шагов через $C'_{\mathrm{E}}(r)$. Таким образом, если $r = \frac{a_0}{a_1}$, то $C'_{\mathrm{E}}(r) = C_{\mathrm{E}}(a_0, a_1)$.

Для дальнейшего будет полезным следующее свойство чисел Фибоначчи:

$$F_n^2 - F_{n-1}F_{n+1} = (-1)^n, \quad n = 1, 2, \ldots \tag{10.6}$$

(доказывается индукцией по n). Рассмотрим вложенные отрезки $\Phi_1 \supset \supset \Phi_2 \supset \Phi_3 \supset \ldots$ числовой прямой:

$$\Phi_n = \left[\frac{F_{2n}}{F_{2n-1}}, \frac{F_{2n+1}}{F_{2n}}\right], \quad n = 1, 2, \ldots$$

Соотношение (10.6) позволяет установить, что ни один из этих отрезков не пуст и что длина отрезка Φ_n равна $\frac{1}{F_{2n-1}F_{2n}}$. Очевидно, что $\Phi_1 = [1, 2]$ (рис. 15). Далее можно доказать, что если рациональное

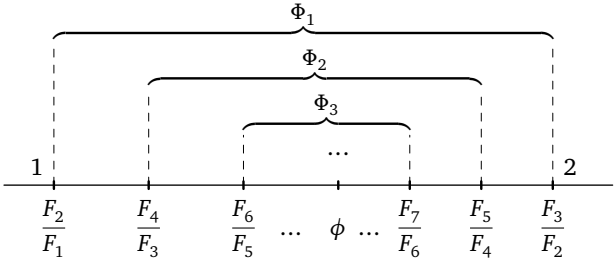


Рис. 15. Расположение чисел $\frac{F_{n+1}}{F_n}$ и интервалов Φ_n, $n = 1, 2, \ldots$

число $\frac{a_0}{a_1}$ принадлежит $\Phi_n, n > 2$, то деление a_0 на a_1 дает ненулевой остаток a_2, и $\frac{a_1}{a_2}$ принадлежит Φ_{n-1}. В самом деле, мы имеем

$$1 < \frac{F_{2n}}{F_{2n-1}} \leqslant \frac{a_0}{a_1} \leqslant \frac{F_{2n+1}}{F_{2n}} < 2.$$

Следовательно, частное от деления a_0 на a_1 с остатком равно 1, т. е. $a_2 = a_0 - a_1$. Далее,

$$\frac{a_0 - a_1}{a_1} = \frac{a_0}{a_1} - 1,$$

поэтому

$$\frac{F_{2n}}{F_{2n-1}} - 1 \leqslant \frac{a_2}{a_1} \leqslant \frac{F_{2n+2}}{F_{2n}} - 1.$$

Но

$$\frac{F_{2n}}{F_{2n-1}} - 1 = \frac{F_{2n} - F_{2n-1}}{F_{2n-1}} = \frac{F_{2n-2}}{F_{2n-1}},$$

и, аналогично, $\frac{F_{2n+1}}{F_{2n}} - 1 = \frac{F_{2n-1}}{F_{2n}}$; мы имеем, следовательно, $\frac{F_{2n-2}}{F_{2n-1}} \leqslant \frac{a_2}{a_1} \leqslant$ $\leqslant \frac{F_{2n-1}}{F_{2n}}$ и

$$\frac{F_{2n}}{F_{2n-1}} \leqslant \frac{a_1}{a_2} \leqslant \frac{F_{2n-1}}{F_{2n-2}}. \tag{10.7}$$

Получаем

$$\frac{a_1}{a_2} \in \left[\frac{F_{2n}}{F_{2n-1}}, \frac{F_{2n-1}}{F_{2n-2}}\right] \subset \left[\frac{F_{2n-2}}{F_{2n-3}}, \frac{F_{2n-1}}{F_{2n-2}}\right] = \Phi_{n-1}.$$

Отрезки $\Phi_2, \Phi_3, \ldots$ не содержат целых чисел, поэтому из доказанного можно заключить, что если $r \in \Phi_n \cap \mathbb{Q}$, $n > 2$, то $C'_{\mathrm{E}}(r) \geqslant n$.

Пусть теперь фиксировано $a_1 \in \mathbb{N}^+$. Рассмотрим множество

$$M = \left\{r : r = \frac{a_0}{a_1}, a_0 = a_1, a_1 + 1, \ldots\right\}.$$

Точки, соответствующие элементам множества M, расположены на правой полуоси, начиная с 1, с шагом $\frac{1}{a_1}$; хотя бы одна из этих точек непременно принадлежит отрезку Φ_n, коль скоро $\frac{1}{a_1}$ меньше длины отрезка Φ_n, т. е. $\frac{1}{a_1} \leqslant \frac{1}{F_{2n-1}F_{2n}}$, или, что то же самое, $a_1 \geqslant F_{2n-1}F_{2n}$. Пусть N — самое большое значение n, для которого выполняется это неравенство (такое N существует, так как единственной точкой, принадлежащей бесконечному числу отрезков рассматриваемого семейства, является $\phi \notin \mathbb{Q}$). Тогда $F_{2N}F_{2N+1} > a_1$. Из формулы Бине следует, что

$$F_{2N}F_{2N+1} = \frac{1}{5}\phi^{4N}(1 - (-\phi)^{-4N})(\phi - (-\phi)^{-4N-1}),$$

откуда

$$F_{2N}F_{2N+1} < c\phi^{4N},$$

где c — некоторая положительная константа. Таким образом, $c\phi^{4N} > > a_1$ и

$$N > \frac{1}{4}\log_\phi a_1 - \log_\phi c.$$

Имеем $C'_{\mathrm{E}}(r) \geqslant N - 1 > \frac{1}{4}\log_\phi a_1 - \log_\phi c - 1$ по крайней мере для одного рационального числа $r \in M$, откуда следует соотношение (10.4).

Переходя к размеру входа $m = \lambda(a_1)$, мы можем воспользоваться теоремой 4.2 из § 4 и получить для сложности $T^*_{\mathrm{E}}(m)$ по числу

делений соотношение $T^*_{\mathrm{E}}(m) = \Theta(m)$. (В силу теоремы 4.2 $T^*_{\mathrm{E}}(m) = \Omega(\log 2^{m-1}) = \Omega(m)$ и $T^*_{\mathrm{E}}(m) = O(\log 2^m) = O(m)$.)

Число шагов расширенного алгоритма Евклида (см. пример 9.1) совпадает с числом шагов обычного алгоритма Евклида. Мы получаем следующее:

Если рассматривать a_1 как размер входа (a_0, a_1) расширенного алгоритма Евклида, то для мультипликативной сложности $T_{\mathrm{EE}}(a_1)$ этого алгоритма выполнено $T_{\mathrm{EE}}(a_1) = \Theta(\log a_1)$. При рассмотрении $m = \lambda(a_1)$ в качестве размера входа a_0, a_1 расширенного алгоритма Евклида имеем для мультипликативной сложности оценку $T^*_{\mathrm{EE}}(m) = \Theta(m)$. Все это сохраняет силу и для сложности $T^*_{\mathrm{E}}(m)$ «обычного» алгоритма Евклида.

Запись алгоритма Евклида в форме (9.1) далека от записи в виде «приличной» компьютерной программы не только в смысле использованной символики, но, главное, в смысле предписываемого расточительного расхода памяти (использован массив, и при буквальном следовании алгоритму (9.1) мы бы имели $S_{\mathrm{E}}(a_1) = \Theta(\log a_1)$). С точки зрения программирования более приемлемой будет, например, запись

```
a := a_0; b := a_1;
while b > 0 do
    d := rem(a, b); a := b; b := d
od
```

(*rem* — знак операции нахождения остатка). После выполнения алгоритма имеем $a = \text{нод}(a_0, a_1)$. Использовано только три дополнительных переменных (a, b и d), и мы получаем $S_{\mathrm{E}}(a_1) = 3$.

Оценку (10.4) можно усилить, показав, что найдется константа γ' такая, что

$$T_{\mathrm{E}}(a_1) > \frac{1}{2}\log_\phi a_1 + \gamma' \tag{10.8}$$

(см. задачу 54). Часто в литературе оценки снизу для сложности в худшем случае алгоритма Евклида по числу делений выводят из оценки, полученной в 1970 г. Г. Хейлбронном для сложности в среднем:

$$\overline{T}_{\mathrm{E}}(a_1) = \frac{12\ln 2}{\pi^2}\ln a_1 + O(1). \tag{10.9}$$

Последняя оценка обосновывается весьма непросто. Из (10.9) выводится, что

$$T_{\mathrm{E}}(a_1) > \frac{12\ln 2}{\pi^2}\ln a_1 + \gamma''$$

для некоторой константы γ''. Однако при больших a_1 оценка (10.8) для $T_{\mathrm{E}}(a_1)$ является более строгой ($\frac{12\ln 2}{\pi^2}\ln\phi = 0{,}40554\ldots < \frac{1}{2}$) и вдобавок выводится элементарно [1].

Может быть доказано (см. задачу 52), что $T_{\mathrm{E}}(a_1) < \log_\phi a_1 + c$, где c — некоторая константа (заметим, что $\log_\phi 2 = 1{,}44\ldots < 2$) [2].

Рассуждения, сходные с приведенными при доказательстве неравенства (10.4), показывают, что для каждого a_0 можно подобрать a_1, $a_0 \geqslant a_1$, так, что будет выполняться

$$C_{\mathrm{E}}(a_0, a_1) > \frac{1}{4}\log_\phi a_0 + \beta, \tag{10.10}$$

где β — некоторая константа. В данном случае можно рассмотреть вложенные отрезки $\Psi_1 \supset \Psi_2 \supset \Psi_3 \supset \ldots$:

$$\Psi_n = \left[\frac{F_{2n}}{F_{2n+1}}, \frac{F_{2n-1}}{F_{2n}}\right], \quad n = 1, 2, \ldots$$

(при $n > 1$ отрезок Ψ_n не содержит чисел вида $\frac{1}{m}$, $m \in \mathbb{N}^+$); если a_0 не делится нацело на a_1 и $\frac{a_1}{a_0} \in \Psi_n$, то $\frac{a_2}{a_1} \in \Psi_{n-1}$ и т. д. Мы воспользуемся этим в § 21.

Пример 10.2. Сортировка массива длины n бинарными вставками выполняется за $n-1$ шаг, причем на l-м шаге число сравнений не превосходит $\lceil\log_2(l+1)\rceil$ и, следовательно, не превосходит $\lceil\log_2 n\rceil$ на любом из шагов, — это приводит к оценке (9.14). Возникает вопрос, не сможем ли мы получить существенно лучшей оценки сверху, рассмотрев сумму $\sum\limits_{l=0}^{n-1}\lceil\log_2(l+1)\rceil$, которая совпадает со значением $T_{\mathrm{B}}(n)$, т. е. сложности по числу сравнений (такое число сравнений достигается, например, в случае, когда изначально массив имеет обратную упорядоченность: $x_1 > x_2 > \ldots > x_n$). Замена переменной суммирования дает

$$T_{\mathrm{B}}(n) = \sum_{k=1}^{n}\lceil\log_2 k\rceil. \tag{10.11}$$

[1] Приведенный выше и дополненный в указании к задаче 54 элементарный вывод оценки (10.8) принадлежит М. Оналбекову; в [19, разд. 4.5.3, упр. 38] со ссылкой на статью Я. Микусинского (1954 г.) приведена оценка, фактически совпадающая с (10.8), но ее вывод опирается на теорию цепных дробей и не столь элементарен и нагляден.

[2] Предположение о том, что $T_{\mathrm{E}}(a_1) \sim \log_\phi a_1$, насколько известно автору, не доказано и не опровергнуто (2008 г.).

Для получения простой асимптотической оценки $T_{\mathrm{B}}(n)$ воспользуемся формулой Стирлинга $n! \sim n^n e^{-n}\sqrt{2\pi n}$, или, что то же самое,

$$n! = n^n e^{-n}\sqrt{2\pi n}(1+o(1)),$$

которая после логарифмирования принимает вид

$$\log_2 n! = n\log_2 n - n\log_2 e + \frac{1}{2}\log_2 n + \frac{1}{2}\log_2 \pi + \frac{1}{2} + o(1)$$

(принято во внимание, что $\log_2(1+o(1)) = o(1)$). Так как $1 < \log_2 e < 2$, то

$$n\log_2 n - 2n < \log_2 n! < n\log_2 n - n \tag{10.12}$$

для всех достаточно больших n; в частности, это означает, что

$$\log_2 n! = n\log_2 n + O(n).$$

Имеем

$$\sum_{k=1}^{n} \lceil \log_2 k \rceil = \sum_{k=1}^{n} \log_2 k + O(n) = \log_2 n! + O(n) = n\log_2 n + O(n).$$

Отсюда получаем оценку

$$T_{\mathrm{B}}(n) = n\log_2 n + O(n), \tag{10.13}$$

из которой следует, в частности, что $T_{\mathrm{B}}(n) \sim n\log_2 n$.

Таким образом, тщательный анализ суммы (10.11) не приводит к существенно лучшей, чем (9.14), оценке, но зато дает асимптотику $T_{\mathrm{B}}(n)$.

В § 14 будет показано, что из того, что сложность по числу сравнений некоторой сортировки не превосходит $n\log_2 n + cn$, где c — некоторая константа, следует, что эта сложность есть $n\log_2 n + O(n)$. Из этого будет следовать, что для сложности по числу сравнений сортировки фон Неймана выполнено $T_{\mathrm{vN}}(n) = n\log_2 n + O(n)$; мы обозначаем эту сортировку буквами vN, от фамилии von Neumann.

§ 11. Завершимость работы алгоритма

Если выполнение циклического (итерационного) алгоритма не завершается для некоторого входа v, то сложность этого алгоритма не определена для значения аргумента, равного $\|v\|$. Поэтому анализ сложности алгоритма прямо или косвенно включает исследование завершимости.

Установление завершимости может быть рассмотрено и как самостоятельная задача. Когда функция L подбирается для выяснения зависимости числа шагов алгоритма от размера входа задачи, то при

наличии выбора более предпочтительной является функция с медленным ростом (имеется в виду рост при увеличении размера входа); даже если при этом усложняется доказательство того, что L обладает всеми нужными свойствами, мы зато получаем более точную оценку числа шагов алгоритма. Но иногда речь может идти только о доказательстве завершимости работы алгоритма — тогда характер роста функции L отходит на второй план. То, что выполнение алгоритма Евклида завершается, было доказано в самом начале обсуждения этого алгоритма без обращения к функциям с логарифмическим ростом.

Пример 11.1. Алгоритм, заданный оператором цикла

$$\texttt{while } n>3 \texttt{ do}\quad \texttt{if } 2\,|\,n \texttt{ then } n:=\frac{n}{2}+1 \texttt{ else } n:=n+1 \texttt{ fi}\quad \texttt{od}$$

(запись $2\,|\,n$ означает, что n делится на 2) завершает свою работу для любого натурального n. Для доказательства достаточно рассмотреть функцию $n-(-1)^n$ и убедиться в ее убывании при $n>3$ в ходе выполнения алгоритма.

Пример 11.2. Если n — данное неотрицательное целое число, то завершимость выполнения алгоритма

$$s:=0;\quad \texttt{for } i=1 \texttt{ to } n \texttt{ do } s:=s+\frac{1}{i} \texttt{ od}$$

очевидна — выполнение оператора цикла завершится после n шагов; легко видеть, что при выполнении этого оператора значение $L(n,i)=$ $=n-i$ убывает, оставаясь при этом неотрицательным целым.

В доказательствах завершимости, основанных на подборе подходящей убывающей целочисленной функции, используется следующее свойство множества $\mathbb{N}$ неотрицательных целых чисел:

Не существует бесконечной убывающей последовательности $n_0>$ $>n_1>n_2>\ldots$ *элементов множества* $\mathbb{N}$.

Имеются и другие примеры (частично) упорядоченных множеств с этим свойством, и соответствующие порядки используются для доказательства завершимости выполнения алгоритмов[1].

[1] Вместо сложных примеров, мы адресуем читателя к задачам 67, 68, заимствованным из [9, разд. 2.3]. В книге [9] кроме двух названных задач содержится интересный и полезный материал о частично упорядоченных множествах и некоторых специальных типах таких множеств. (Решения задач 67, 68 не обязательно должны содержать явные упоминания порядков на множествах, хотя введение некоторых порядков полностью проясняет ситуацию.)

Установление завершимости выполнения алгоритма иногда оказывается очень трудной задачей.

Пример 11.3. Завершимость для любого натурального n выполнения алгоритма

```
while n > 1 do
   if 2 | n then n := n/2 else n := 3n + 1 fi
od
```

не доказана и не опровергнута по сей день, хотя на это направлялись серьезные усилия (см. [53]).

В случае рандомизированных алгоритмов мы сталкиваемся (см. ниже пример 11.4 и первую часть примера 11.5) с возможностью того, что алгоритм завершается с вероятностью 1, но математическое ожидание времени от начала выполнения до полного завершения алгоритма бесконечно. Поэтому утверждения о завершимости возможны в двух формах: завершимость с вероятностью 1 и конечность ожидаемого времени выполнения. Вторая форма является, вообще говоря, более сильной.

Пример 11.4. В теории вероятностей подробно рассматривается задача о блуждании частицы по прямой: за один шаг частица передвигается на 1 вправо или на то же расстояние влево, направление же движения на каждом шаге выбирается случайно, вероятности выбора каждого из направлений одинаковы и равны $\frac{1}{2}$. Пусть в начальный момент частица находится в точке 0, и на прямой отмечена некоторая точка m с целой координатой. Доказывается, что с вероятностью 1 частица через некоторое время попадает в точку m. Попытаемся теперь вычислить математическое ожидание a времени (общего числа шагов), которое потребуется для достижения точки m. Легко убедиться, что уже в случае $m = \pm 1$ это среднее бесконечно. Пусть, например, $m = 1$. Используем формулу полного математического ожидания: если шаг сделан вправо, то за этот один шаг мы достигаем цели, если же шаг сделан влево, то придется дождаться момента, когда частица вернется в точку 0 и тем самым повторится начальная ситуация:

$$a = 1 \cdot \frac{1}{2} + 2a \cdot \frac{1}{2}.$$

Если бы a было конечным, это бы дало $0 = \frac{1}{2}$. Поэтому среднее бесконечно.

Рассмотрим с этой точки зрения задачу 17. Итак, путник столкнулся со стеной, простирающейся бесконечно в обе стороны. Имеется дверь в этой стене, но путник не знает ни расстояния до двери, ни направления к ней. Если путник использует алгоритм поиска двери, состоящий в бросании перед каждым шагом монеты для определения направления (вправо или влево) этого шага, то он с вероятностью 1 найдет дверь, но математическое ожидание числа шагов равно бесконечности, коль скоро в начальный момент путник не стоит прямо перед дверью. Иными словами, если p_k $(k \geqslant 1)$ есть вероятность того, что после k шагов путник впервые окажется перед дверью, то

$$\sum_{k=1}^{\infty} p_k = 1 \quad \text{и} \quad \sum_{k=1}^{\infty} k p_k = \infty.$$

Пример 11.5. Обратимся к системам автоматизированного обучения. При использовании известного педагогического приема — задания вопросов вразбивку — с каждым выбранным системой вопросом может быть связана следующая цепочка действий: вопрос обучаемому; прием и проверка ответа; сообщение, правилен ли ответ и объявление правильного ответа, если был дан неправильный ответ. Нетрудно привести примеры учебных тем, когда такой подход выглядит разумно: таблица умножения, исторические даты, иностранные слова (здесь же — неправильные глаголы), значения иероглифов, название созвездий (показываются картинки), определение на слух музыкальных интервалов и т. д. Прообразом одного из алгоритмов выбора вопросов служит способ заучивания ответов с помощью колоды карточек, на лицевой стороне каждой из которых написан вопрос, а на обороте — ответ. Из колоды выбирается наугад карточка, и делается попытка ответить на вопрос. Если это не удается, ответ прочитывается на обороте. Затем карточка возвращается в колоду, и все повторяется. На примере колоды карточек предлагаемый рандомизированный алгоритм (называемый алгоритмом кратных карт [1]) может быть проинтерпретирован так: выбранная карточка, содержащая вопрос с известным обучаемому ответом, уже не возвращается в колоду; если же обучаемый не смог дать правильный ответ, то, кроме того что в колоду возвращается эта карточка, туда добавляется еще один ее экземпляр. Сеанс обучения заканчивается, когда в колоде не остается карточек.

Пусть вопросы занумерованы числами $1, 2, \ldots, n$. Каждый этап сеанса обучения характеризуется кратностями $m_1, m_2, \ldots, m_n$ вопро-

[1] См. [3].

сов (не исключается равенство нулю каких-то кратностей); пусть $m = m_1 + m_2 + \ldots + m_n$. Если $m = 0$, то сеанс заканчивается, иначе выбирается очередной вопрос; вероятность того, что будет выбран i-й вопрос, должна равняться $\frac{m_i}{m}$.

Сеанс обучения становится бесконечным, если, например, с некоторого момента обучаемый начинает давать только неправильные ответы. Но можно интересоваться вероятностями и средним временем ожидания некоторых событий в течение этого бесконечного сеанса. Пусть в некоторый момент только один вопрос, скажем, вопрос с номером 1, имеет кратность 1, а все остальные — кратности большие, чем 1. Предположим, что начиная с этого момента обучаемый стабильно дает неправильные ответы на предлагаемые вопросы. Найдем вероятность того, что рано или поздно обучаемый получит вопрос с номером 1, а также математическое ожидание времени, проходящего до появления этого вопроса (время измеряется количеством заданных вопросов). Пусть m — суммарная кратность, а n — общее число вопросов, $m > n \geqslant 2$. Вероятность получить вопрос с номером 1 после i вопросов с другими номерами равна $\frac{1}{m}$, если $i = 0$, и равна

$$\frac{m-1}{m} \cdot \frac{m}{m+1} \cdots \frac{m+i-2}{m+i-1} \cdot \frac{1}{m+i} = \frac{m-1}{(m+i-1)(m+i)},$$

если $i \geqslant 1$. Поэтому вероятность получить рано или поздно вопрос с номером 1 равна

$$\frac{1}{m} + (m-1) \sum_{i=1}^{\infty} \frac{1}{(m+i-1)(m+i)}. \tag{11.1}$$

Так как

$$\frac{1}{(m+i-1)(m+i)} = \frac{1}{m+i-1} - \frac{1}{m+i},$$

то для частичной суммы

$$s_N = \sum_{i=1}^{N} \frac{1}{(m+i-1)(m+i)}$$

имеем

$$s_N = \frac{1}{m} - \frac{1}{m+N},$$

и $\lim\limits_{n\to\infty} s_N = \frac{1}{m}$, откуда значение выражения (11.1) (искомая вероятность) есть

$$\frac{1}{m} + \frac{m-1}{m} = 1.$$

Математическое ожидание времени есть

$$(m-1)\sum_{i=1}^{\infty}\frac{i}{(m+i-1)(m+i)}. \tag{11.2}$$

При этом ряд $\sum\limits_{i=1}^{\infty}\frac{i}{(m+i-1)(m+i)}$ расходится: члены этого ряда положительны, и i-й член есть $\Theta\left(\frac{1}{i}\right)$ (является величиной порядка $\frac{1}{i}$). Отсюда математическое ожидание (11.2) равно бесконечности.

Пусть теперь u вопросов (будем считать, что это вопросы с номерами $1, 2, \ldots, u$) имеют кратность 1, а остальные $n-u$ вопросов — кратность большую, чем 1, и пусть $u \geqslant 2$. Оказывается, что при стабильных неправильных ответах обучаемого на задаваемые вопросы математическое ожидание времени, проходящего до получения всех, кроме какого-то одного, вопросов с номерами $1, 2, \ldots, u$, есть конечная величина. Докажем конечность среднего времени ожидания какого-нибудь одного вопроса с номером от 1 до u, далее можно применить индукцию. Вероятность получить такого рода вопрос после i вопросов с номерами из диапазона от $u+1$ до n полностью определяется значениями u, m, i:

$$\frac{m-u}{m}\cdot\frac{m-u+1}{m+1}\cdots\frac{m-u+i-1}{m+i-1}\cdot\frac{u}{m+i}.$$

Эта вероятность при $i > u$ равна

$$\frac{u(m-u)(m-u+1)\ldots(m-1)}{(m-u+i)(m-u+i+1)\ldots(m+i-1)(m+i)}.$$

Поэтому искомое математическое ожидание есть сумма некоторого конечного числа слагаемых и ряда

$$\sum_{i=u+1}^{\infty}\frac{u(m-u)(m-u+1)\ldots(m-1)}{(m-u+i)(m-u+i+1)\ldots(m+i-1)(m+i)}(i+1),$$

или

$$u(m-u)(m-u+1)\ldots(m-1)\sum_{i=u+1}^{\infty}\frac{i+1}{(m-u+i)(m-u+i+1)\ldots(m+i)}.$$

Упрощая, получаем

$$u(m-u)(m-u+1)\ldots(m-1)\sum_{j=1}^{\infty}\frac{j+u+1}{(j+m)(j+m+1)\ldots(j+m+u)}.$$

Ряд, входящий в это выражение, сходится, так как j-й член ряда есть $\Theta\left(\frac{1}{j^u}\right)$ и $u \geqslant 2$.

Это рассуждение сохраняет силу и в том случае, когда вопросы с номерами $1, 2, \ldots, u$ являются одним и тем же вопросом, что позволяет отметить следующую характерную черту алгоритма кратных карт:

Если изначально все вопросы имеют кратность 1 *и обучаемый дает на все вопросы неправильные ответы, то математическое ожидание времени до наступления момента, к которому обучаемый получит хотя бы по одному разу каждый из вопросов, равно бесконечности, хотя вероятность наступления такого момента равна* 1. *Если же изначально все вопросы имели большие единицы кратности (например, все они имели кратность* 2*), то обсуждаемое математическое ожидание конечно.*

§ 12. Вложенные циклы (дополнительные примеры)

При анализе сложности вложенных циклов мы естественно приходим к вычислению и оцениванию кратных сумм.

Пример 12.1. Число обменов при транспонировании $(n \times n)$-матрицы (a_{ij})

```
for i = 1 to n − 1 do
   for j = i + 1 to n do a_ij ↔ a_ji od
od
```

есть $\sum\limits_{i=1}^{n-1} \sum\limits_{j=i+1}^{n} 1 = \sum\limits_{i=1}^{n-1} (n-i) = \frac{n^2-n}{2}$. Аналогично, распознавание симметричности данной $(n \times n)$-матрицы требует в худшем случае сравнений в количестве $\sum\limits_{i=1}^{n-1} \sum\limits_{j=i}^{n} 1 = \sum\limits_{i=1}^{n-1} (n-i+1) = \frac{(n+1)(n-2)}{2}$ и т. д.

Легкое получение ответов в примере 12.1 объясняется тем, что в выписанных формальным путем суммах нижние границы суммирования не превосходили верхних; в таких случаях всегда, например, $\sum\limits_{j=p}^{q} 1 = q - p + 1$. Но, получая сумму этим формальным путем из оператора цикла, можно столкнуться с тем, что $p > q$ (оператор цикла затратит 0 шагов), и ответ $q - p + 1$ будет неправильным. Соответствующий пример дается в задаче 71.

Подходя к асимптотическим оценкам, отметим один простой, но полезный факт.

Предложение 12.1. *Пусть Λ — один из символов O, Θ, Ω. Пусть $a(k)$, $b(k)$ определены для всех $k \in \mathbb{N}^+$ и принимают положительные значения (не обязательно целые); пусть $a(k) = \Lambda(b(k))$, $k \to \infty$. Тогда $\sum\limits_{k=1}^{n} a(k) = \Lambda\Big(\sum\limits_{k=1}^{n} b(k)\Big)$, $n \to \infty$. Утверждение остается справедливым и при замене верхней границы суммирования n на $\varphi(n)$ для любой функции $\varphi(n)$, определенной для всех $n \in \mathbb{N}^+$ и принимающей значения в $\mathbb{N}^+$.*

Доказательство. Докажем первую часть утверждения для $\Lambda = \Theta$, остальные случаи доказываются аналогично. В силу замечания, сделанного в § 2 после предложения 2.1,

$$c_1 b(k) \leqslant a(k) \leqslant c_2 b(k)$$

для всех $k \in \mathbb{N}^+$ и некоторых $c_1, c_2 > 0$. Отсюда

$$c_1 \sum_{k=1}^{n} b(k) \leqslant \sum_{k=1}^{n} a(k) \leqslant c_2 \sum_{k=1}^{n} b(k). \qquad \square$$

Пример 12.2. Проведем асимптотический анализ следующего алгоритма, оценивая общее число выполняемых операций в зависимости от исходного значения n:

$l := 0$;
`for` $i = 1$ `to` $\lfloor \sqrt{n^3 + 1} \rfloor$ `do`
  $k := i$;
  `while` $k > 1$ `do` $l := l + k$; $k := \left\lfloor \frac{k}{3} \right\rfloor$ `od`
`od`

Для внутреннего цикла, выполняемого при начальном значении k, равном значению i, число шагов и общее число операций допускают оценку $\Theta(\log i)$ (можно представить себе, что k записано в троичной системе счисления, тогда каждый шаг удаляет одну цифру из записи k). Таким образом, затраты на выполнение внешнего цикла представляют собой $\sum\limits_{k=1}^{\varphi(n)} a(k)$, где $\varphi(n) = \lfloor \sqrt{n^3 + 1} \rfloor$, $a(k) = \Theta(\log k)$, и по предложению 12.1 допускают оценку $\Theta\Big(\sum\limits_{k=1}^{\varphi(n)} \log k\Big)$ или $\Theta(\log(\varphi(n)!)$. Так как $\varphi(n) \to \infty$ при $n \to \infty$, то можно применить соотношение $\log(\varphi(n)!) = \Theta(\varphi(n) \log \varphi(n))$, которое являет-

ся следствием формулы Стирлинга (см. § 9). Это дает нам оценку $\Theta(\lfloor\sqrt{n^3+1}\rfloor \log\lfloor\sqrt{n^3+1}\rfloor)$ для интересующих нас затрат. После упрощения получаем $\Theta(n^{3/2}\log n)$. В этом примере затраты однозначно определяются по n, поэтому полученная оценка определяет асимптотику сложности рассматриваемого алгоритма при использовании n в качестве размера входа.

Оценку $O(\lfloor\sqrt{n^3+1}\rfloor \log\lfloor\sqrt{n^3+1}\rfloor)$ и, тем самым, $O(n^{3/2}\log n)$ можно было бы получить не прибегая к формуле Стирлинга, исходя из того, что сумма конечного числа неотрицательных слагаемых не превосходит произведения наибольшего слагаемого на число слагаемых суммы. Для Θ этот прием не работает, как показывает пример суммы $\sum\limits_{i=1}^{n} 2^i$.

§ 13. Нецелые размеры входа и непрерывные оценочные функции

Предложение 13.1. *Пусть для некоторого алгоритма A можно указать входы $v_0, v_1, \ldots$, размеры которых ограничены сверху и снизу положительными константами c_1 и c_2:*

$$c_1 \leqslant \|v_i\| \leqslant c_2, \quad i = 0, 1, \ldots,$$

и в то же время затраты $C_A^T(v_i)$ неограниченно возрастают при $i \to \infty$. Тогда не существует такой непрерывной на отрезке $[c_1, c_2]$ функции $f(x)$ вещественной переменной, что $T_A(x) \leqslant f(x)$ для всех значений x размера входа, принадлежащих отрезку $[c_1, c_2]$.

Доказательство. Очевидно, что $T_A(\|v_i\|) \geqslant C_A^T(v_i)$, и поэтому последовательность $T_A(\|v_i\|)$, $i = 0, 1, \ldots$, не ограничена. В то же время, любая функция f, непрерывная на $[c_1, c_2]$, ограничена на этом отрезке. □

Сформулируем еще одно столь же легко доказываемое утверждение (доказательство опускаем).

Предложение 13.2. *Пусть A — некоторый алгоритм, и $f(x)$ — функция вещественной переменной, определенная на некотором интервале I. Пусть $T_A(x) \leqslant f(x)$ всякий раз, когда $x \in I$ и значение $T_A(x)$ определено. Пусть $x_0 \in I$ и для любых $\varepsilon > 0$, $N > 0$ найдется такой вход v алгоритма A, что $|x_0 - \|v\|| < \varepsilon$ и $C_A^T(v) > N$. Тогда $f(x)$ разрывна в точке x_0.*

Пример 13.1. Вновь рассмотрим алгоритм Евклида, используя теперь в качестве размера входа (a_0, a_1) рациональное число $\frac{a_0}{a_1}$, которое, как указывалось в примере 9.1, однозначно определяет соответствующее количество делений с остатком (сложность $T'_{\mathrm{E}}\left(\frac{a_0}{a_1}\right)$, соответствующая этому размеру, совпадает с затратами: $C_{\mathrm{E}}(a_0, a_1)$). В силу формулы Бине (10.2) имеем

$$\lim_{n\to\infty} \frac{F_{n+1}}{F_n} = \phi,$$

при этом $C_{\mathrm{E}}(F_{n+1}, F_n) = n$. Поэтому если функция вещественной переменной $f(x)$ такова, что $C_{\mathrm{E}}(a_0, a_1) \leqslant f\left(\frac{a_0}{a_1}\right)$, то, в силу предложения 13.2, $f(x)$ разрывна в точке ϕ. Мы воспользовались тем, что для любого $\varepsilon > 0$ и любого натурального N найдется рациональное число u/v такое, что $|\phi - \frac{u}{v}| < \varepsilon$ и $C_{\mathrm{E}}(u, v) \geqslant N$.

Можно доказать, что здесь дело не в числе ϕ — то же самое справедливо для любого $x_0 > 1$. Пусть $0 < \varepsilon < x_0 - 1$. Докажем, что в рациональных точках ε-окрестности $V_{x_0,\varepsilon}$ числа x_0 функция $T'_{\mathrm{E}}(r)$ не является ограниченной. Предположим противное: пусть $r = \frac{u_0}{u_1} \in V_{x_0,\varepsilon}$ — рациональное число, на котором достигается максимум. Пусть частные, возникающие в процессе применения алгоритма Евклида к u_0, u_1, равны $q_1, q_2, \ldots, q_n$:

$$u_{i-1} = q_i u_i + u_{i+1}, \quad i = 1, 2, \ldots, n, \quad u_{n+1} = 0.$$

Выберем $\delta > 0$ столь малым, что $V_{r,\delta} \subset V_{x_0,\varepsilon}$, и возьмем любое $s \in \mathbb{Q} \setminus \mathbb{Z}$ такое, что $\left|\frac{u_{n-1}}{u_n} - s\right| < \delta$ (легко заметить, что $\frac{u_{n-1}}{u_n} = q_n \in \mathbb{Z}$). Пусть l, m — такие целые, что $s = \frac{l}{m}$. Используя $q_1, q_2, \ldots, q_n$, найдем $v_0, v_1, \ldots$ $\ldots, v_n \in \mathbb{N}^+$, для которых

$$v_{i-1} = q_i v_i + v_{i+1}, \quad i = 1, 2, \ldots, n,$$

и $v_{n-1} = l$, $v_n = m$. Индукцией по k легко доказать, что для $k = 0, 1, \ldots$ $\ldots, n-1$

$$\left|\frac{v_{n-k-1}}{v_{n-k}} - \frac{u_{n-k-1}}{u_{n-k}}\right| < \delta. \tag{13.1}$$

В самом деле, для $k = 0$ это очевидно; остается показать, что при $0 < k < n-1$ из (13.1) следует

$$\left|\frac{v_{n-k-2}}{v_{n-k-1}} - \frac{u_{n-k-2}}{u_{n-k-1}}\right| < \delta. \tag{13.2}$$

Имеем

$$\frac{v_{n-k-2}}{v_{n-k-1}} = \frac{q_{n-k-1}v_{n-k-1}+v_{n-k}}{v_{n-k}} = q_{n-k-1} + \frac{v_{n-k}}{v_{n-k-1}},$$
$$\frac{u_{n-k-2}}{u_{n-k-1}} = \frac{q_{n-k-1}u_{n-k-1}+u_{n-k}}{u_{n-k}} = q_{n-k-1} + \frac{u_{n-k}}{u_{n-k-1}},$$

откуда

$$\left|\frac{v_{n-k-2}}{v_{n-k-1}} - \frac{u_{n-k-2}}{u_{n-k-1}}\right| = \left|\frac{v_{n-k}}{v_{n-k-1}} - \frac{u_{n-k}}{u_{n-k-1}}\right| = \frac{|u_{n-k-1}v_{n-k} - u_{n-k}v_{n-k-1}|}{u_{n-k-1}v_{n-k-1}}.$$

Неравенство (13.1) можно переписать в виде

$$\frac{|u_{n-k-1}v_{n-k} - u_{n-k}v_{n-k-1}|}{u_{n-k}v_{n-k}} < \delta;$$

это неравенство влечет

$$\frac{|u_{n-k-1}v_{n-k} - u_{n-k}v_{n-k-1}|}{u_{n-k-1}v_{n-k-1}} < \delta,$$

так как $u_{n-k-1}v_{n-k-1} > u_{n-k}v_{n-k}$. Неравенство (13.2) доказано, и индукция завершена. Следовательно, $\frac{v_0}{v_1} \in V_{s,\delta}$ и, тем самым, $\frac{v_0}{v_1} \in V_{x_0,\varepsilon}$. Но $T'_{\mathrm{E}}\left(\frac{v_0}{v_1}\right) > n$, так как $s = \frac{v_{n-1}}{v_n} \notin \mathbb{Z}$. Это противоречит предположению о максимальности значения $T'_{\mathrm{E}}\left(\frac{u_0}{u_1}\right)$.

В силу предложения 13.2 мы получаем следующее:

Пусть размер каждого входа (a_0, a_1) алгоритма Евклида определен как $r = a_0/a_1$ (рациональное число $\geqslant 1$), и в соответствии с этим рассматривается сложность $T'_{\mathrm{E}}(r)$ этого алгоритма по числу делений. Пусть функция $f(x)$ вещественной переменной, определенная всюду на бесконечном полуинтервале $I = [1, \infty)$, такова, что $T'_{\mathrm{E}}(r) < f(r)$ при всех рациональных $r \geqslant 1$. Тогда $f(x)$ является разрывной в любой принадлежащей I точке.

Нелишне заметить, что при целочисленном размере входа условие предложения 13.1 может выполняться, если только для некоторого целого n, $c_1 \leqslant n \leqslant c_2$, сложность $T_A(n)$ бесконечна.

Приводившиеся ранее оценки (9.15), (9.16) для числа итераций методов деления пополам и касательных используют, фактически, нецелый размер $1/\varepsilon$. Положение спасается тем, что при малом изменении ε не происходит катастрофического увеличения числа итераций. Эту ситуацию можно было бы считать типичной, а пример 13.1 искусственным, если бы упомянутые возможные трудности не приводили бы иногда к реальным ошибкам. В приложении С рассмотрена известная алгоритмическая проблема, связанная с алгебраическими

числами («проблема орбит»), для которой в свое время было предложено и опубликовано неверное решение, и причина ошибки была в том, что авторы проглядели ту негативную возможность, которая показана в примере 13.1.

В ситуациях, подобных описанной в примере 13.1, мы не можем использовать в оценках вида $T_A(x) < f(x)$, $T_A(x) = O(f(x))$, $T_A(x) = \Theta(f(x))$ в качестве функции $f(x)$, скажем, логарифм, степенную и показательную функции, полиномы и т. д. Во многих случаях это означает, что размер как функция входа выбран неудачно. В качестве размера входа алгоритма надо, по возможности, выбирать такие величины, исходя из которых сложность алгоритма может быть выражена или оценена с помощью простых аналитических функций. Это позволит составить представление об изменении сложности при возрастании размера входа. Появление функций, поведение которых трудно себе вообразить, лишает возможности получения такой информации.

Задачи

48. Функцию $L(s)$ не обязательно выбирать целочисленной. Достаточно, чтобы ее значения были неотрицательными и каждый шаг алгоритма понижал ее значение на величину, ограниченную снизу некоторым положительным числом d. Как с помощью такой функции оценить число шагов?

49. При использовании $m = \lambda(a_1)$ в качестве размера входа алгоритма Евклида и его расширенного варианта для соответствующих мультипликативных сложностей справедливы верхние оценки $2m - 1$ и, соответственно, $6m + 3$.

50. Бинарный алгоритм Евклида основан на том, что если a_0 и a_1 четны, то $\text{нод}(a_0, a_1) = 2\,\text{нод}\left(\frac{a_0}{2}, \frac{a_1}{2}\right)$; если a_0 четно, но a_1 нечетно, то $\text{нод}(a_0, a_1) = \text{нод}\left(\frac{a_0}{2}, a_1\right)$; если a_0 нечетно, но a_1 четно, то $\text{нод}(a_0, a_1) = \text{нод}\left(a_0, \frac{a_1}{2}\right)$; если a_0 и a_1 нечетны, то можно перейти к вычислению $\text{нод}(a_0 - a_1, a_1)$ при $a_0 \geqslant a_1$ и к вычислению $\text{нод}(a_0, a_1 - a_0)$ при $a_0 < a_1$. Вычисления заканчиваются, как только на некотором этапе вычитание даст нуль. Выполнение этого алгоритма завершается для любых исходных a_0, a_1, для которых $a_0 \geqslant a_1 > 0$. Для числа шагов алгоритма как функции от a_0, a_1 справедлива оценка $O(\log a_0)$.

Указание. По крайней мере на одном из двух последовательных шагов этого алгоритма по крайней мере одно из чисел делится на два.

51. Пусть в ходе применения алгоритма Евклида к числам a_0, a_1, $a_0 > a_1$, возникают ненулевые остатки $a_2, a_3, \ldots, a_n$, и $a_{n+1} = 0$. Тогда $a_{n-t} \geqslant F_{t+2}$, $t = 0, 1, \ldots, n-1$. Отсюда следует справедливость следующего предложения, обычно именуемого в литературе теоремой Ламе. Пусть в ходе применения алгоритма Евклида к числам a_0, a_1, $a_0 > a_1$, возникают ненулевые остатки $a_2, a_3, \ldots, a_n$. Тогда $a_1 \geqslant F_{n+1}$.

52. (Продолжение задачи 51.) Пусть в ходе применения алгоритма Евклида к числам a_0, a_1, $a_0 > a_1$, возникают ненулевые остатки $a_2, a_3, \ldots, a_n$. Тогда $n < \log_\phi(a_1 + 1) + \log_\phi \sqrt{5} - 1$, и, как следствие, $n < \log_\phi a_1 + c$ для некоторой константы c.

53. Привести доказательство равенства (10.6).

54. Доказать оценку (10.8).

Указание. В § 10 при рассмотрении примера 13.1 уже доказано, что если $\frac{a_0}{a_1} \in \Phi_n$, $n > 2$, то выполнены неравенства (10.7). Вывести отсюда, что $\frac{F_{2n-2}}{F_{2n-3}} \leqslant \leqslant \frac{a_2}{a_3} \leqslant \frac{F_{2n-1}}{F_{2n-2}}$, т. е. $\frac{a_2}{a_3} \in \Phi_{n-1}$, $n > 2$.

55. Разработать алгоритм, распознающий существование у уравнения

$$ax + by = c, \quad a, b, c \in \mathbb{Z}, \tag{13.3}$$

решения в неотрицательных целых числах, и исследовать мультипликативную сложность этого алгоритма.

Указание. Уравнение (13.3) тогда и только тогда имеет решение в целых числах, когда $d \mid c$, где $d = \text{нод}(a, b)$.

56. Если x_0, y_0 — одно из целочисленных решений уравнения (13.3), то все решения описываются формулами

$$x = x_0 + \frac{b}{d}t, \quad y = y_0 - \frac{a}{d}t, \quad t = 0, \pm 1, \pm 2, \ldots,$$

$d = \text{нод}(a, b)$.

Указание. Если x_0, y_0 и x_1, y_1 — два различных решения уравнения (13.3), то $x_0 - x_1, y_0 - y_1$ — решение уравнения $\frac{a}{d}x + \frac{b}{d}y = 0$.

57. Числа s_i, t_i, получаемые в ходе выполнения расширенного алгоритма Евклида, являются взаимно простыми при $i = 0, 1, \ldots, n+1$.

Указание. Индукцией по i доказать равенство

$$\begin{pmatrix} -q_1 & 1 \\ 1 & 0 \end{pmatrix} \ldots \begin{pmatrix} -q_{i-1} & 1 \\ 1 & 0 \end{pmatrix} = \begin{pmatrix} t_i & t_{i-1} \\ s_i & s_{i-1} \end{pmatrix}$$

для $i = 2, 3, \ldots, n-1$ и рассмотреть определители обеих его частей.

58. При некоторых значениях n число сравнений, затрачиваемых при бинарном поиске, не определяется однозначно исходя лишь из значения n (например, зная лишь, что $n=6$, мы не можем указать точное число сравнений). Но существует бесконечно много n таких, для которых это значение определяется единственным образом и равно $\lfloor \log_2 n \rfloor + 1$.

59. Бинарный поиск места элемента y в упорядоченном массиве $x_1 < x_2 < \ldots < x_n$ требует $\lfloor \log_2(n+1) \rfloor$ сравнений в лучшем случае. Как следствие, для значений n вида $2^k - 1$, $k = 1, 2, \ldots$, и только для них число сравнений однозначно определено.

Указание. Пусть $\mu(n)$ — минимум числа сравнений при бинарном поиске места элемента. Имеем $\mu(1) = 1$, и по индукции можно доказать, что при $n \geqslant 2$ выполнено $\mu(n) \geqslant \mu(n-1)$ и $\mu(n) = \mu\left(\left\lfloor \frac{n-1}{2} \right\rfloor\right) + 1$. Отсюда следует, что $\mu(n)$ равно числу положительных элементов последовательности

$$n, \quad \left\lfloor \frac{n-1}{2} \right\rfloor, \quad \left\lfloor \frac{\left\lfloor \frac{n-1}{2} \right\rfloor - 1}{2} \right\rfloor, \quad \ldots \tag{13.4}$$

Заметим, что $\lambda\left(\left\lfloor \frac{n-1}{2} \right\rfloor\right) = \lambda(n) - 1$, если $n \neq 2^k$, и $\lambda\left(\left\lfloor \frac{n-1}{2} \right\rfloor\right) = \lambda(n) - 2$, если $n = 2^k$, $k > 0$. Если $n = 2^k - 1$, $k > 0$, то число положительных элементов (13.4) равно $\lambda(n)$; в остальных случаях в последовательности (13.4) имеется в точности один элемент вида 2^k, $k > 0$, и число положительных элементов рассматриваемой последовательности равно $\lambda(n) - 1$.

60. *Опорная прямая* к многоугольнику P, проходящая через точку Q, внешнюю по отношению к многоугольнику, — это прямая, которой принадлежит по крайней мере одна вершина многоугольника P, и при этом все вершины этого многоугольника лежат в одной полуплоскости по отношению к этой прямой (рис. 16). Предложить имеющий сложность в $O(n)$ по общему числу операций алгоритм проведения двух опорных прямых к данному выпуклому n-угольнику из данной точки.

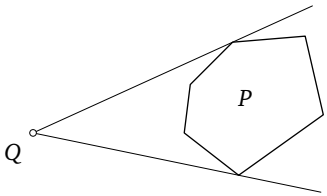


Рис. 16. Опорные прямые к многоугольнику.

Указание. Сначала найдем сторону n-угольника, которая видна из точки Q (см. пример 9.2). Это даст две соседние вершины n-угольника. Через точку Q и каждую из этих вершин проведем прямые. Затем сдвигаемся от каждой из этих двух вершин в сторону, противоположную другой вершине, пока увеличивается угол с вершиной Q.

61. Сложность алгоритма, схематически описанного в указании к предыдущей задаче, есть $\Theta(n)$. (Вместе с тем, возможно построение опорных прямых со сложностью $\Theta(\log n)$ — для этого надо применить некий вариант бинарного поиска для нахождения каждой из тех двух вершин, через которые проходят искомые опорные прямые.)

62. Предложить имеющий сложность $O(n_1 + n_2)$ по общему числу операций алгоритм построения выпуклой оболочки объединения двух выпуклых многоугольников, имеющих, соответственно, n_1 и n_2 вершин, считая $n_1 + n_2$ размером входа.

Указание. Один из известных алгоритмов этого построения основывается на том, что, взяв некоторую точку Q, принадлежащую первому многоугольнику, мы, в том случае, когда эта точка принадлежит и второму многоугольнику, можем путем слияния двух упорядоченных последовательностей получить последовательность всех вершин обоих многоугольников, упорядоченных по величине углов, под которыми видны эти вершины по отношению к какой-нибудь фиксированной прямой, проходящей через Q. После этого, как в алгоритме Грэхема (см. пример 3.1), можно удалить вдавленные вершины. Если Q не принадлежит второму многоугольнику, то проведем из Q две опорные прямые ко второму многоугольнику. Пусть S и T — вершины второго многоугольника, через которые проходят опорные прямые (если опорная прямая проходит через несколько вершин, то из них выбирается наиболее удаленная от Q). Исключим из рассмотрения все принадлежащие треугольнику QST вершины второго многоугольника, кроме вершин S и T (рис. 17). Оставшиеся вершины рассматриваем в порядке возрастания углов; сливаем, как и в первом случае, две упорядоченные последовательности, а затем удаляем вдавленные вершины.

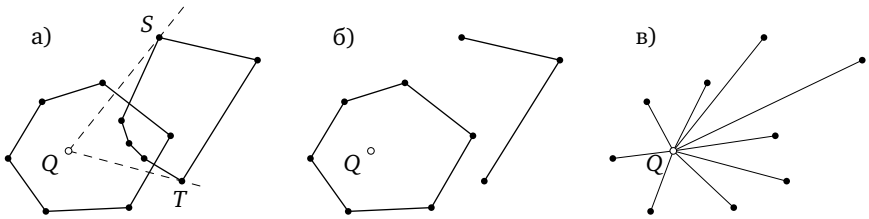


Рис. 17. Этапы построения выпуклой оболочки объединения двух выпуклых многоугольников.

63. Верно ли, что сложность по числу сравнений сортировки массива из n элементов бинарными вставками есть $n \log_2 n + O(1)$?

64. Обратимся к известной школьной задаче: имеются 3^m монет, из которых одна фальшивая (более легкая); с помощью чашечных весов без гирь надо за m взвешиваний определить фальшивую мо-

нету. Сравнение по весу двух групп по 3^{m-1} монет сужает диапазон последующего поиска до 3^{m-1} монет, поэтому в итоге будет затрачено m взвешиваний, как и требуется. Для числа монет $n = 3^m$ мы имеем алгоритм, использующий $\log_3 n$ взвешиваний. (Здесь мы рассматриваем n в качестве размера исходной группы монет.) Можно обобщить этот алгоритм на произвольное число монет n так, что число взвешиваний будет ограничено близкой к $\log_3 n$ функцией: сравниваем по весу две группы, в каждой из которых по $\lceil n/3 \rceil$ монет, это сужает диапазон дальнейшего поиска до не превосходящего $\lceil n/3 \rceil$ числа монет; затем действуем тем же способом. Получить верхнюю границу $\lceil \log_3 n \rceil$ для числа взвешиваний сопоставлением размеров групп с элементами последовательности $1, 3, 9, \ldots$ подобно тому, как при анализе сортировки фон Неймана количество упорядоченных сегментов сопоставлялось с элементами последовательности $1, 2, 4, \ldots$

65. Изменим алгоритм поиска фальшивой монеты (задача 64): будем сравнивать по весу две группы, в каждой из которых по $\lfloor n/3 \rfloor$ монет, и т. д.

а) Показать, что этому алгоритму может потребоваться число взвешиваний, которое превосходит $\lceil \log_3 n \rceil$.

б) Получить верхнюю границу $\lfloor \log_3 n \rfloor + 2$ для числа взвешиваний сопоставлением размеров групп с элементами последовательности $1 + 2, 3 + 2, 9 + 2, \ldots$ подобно тому, как в при анализе бинарного поиска размеры сегментов сопоставлялись с элементами последовательности $1, 2, 4, \ldots$

66. Вернемся к начальному варианту алгоритма определения фальшивой монеты (задача 64). При некоторых значениях n количество взвешиваний не определяется однозначно исходя лишь из значения n (например, при $n = 10$), хотя и существует бесконечно много n таких, для которых это значение определяется единственным образом.

67. Каждый день бизнесмен дает черту одну монету, и в обмен получает любой набор монет по своему выбору, но меньшего достоинства. Видов монет конечное число; менять или получать деньги в другом месте бизнесмен по условиям сделки не может. Если у бизнесмена не остается монет достоинством выше минимального, он проигрывает. На таких условиях рано или поздно бизнесмен терпит поражение, каков бы ни был первоначальный запас его монет.

68. Имеется конечная последовательность нулей и единиц. Разрешается найти в ней группу 01 и заменить на 100…00 (при этом можно написать сколько угодно нулей). Доказать, что это преобразование не может быть произведено бесконечно много раз.

69. Алгоритм, приведенный в примере 11.1, затрачивает $O(\log n)$ шагов.

70. Вернемся к алгоритму кратных карт (пример 11.5). Если изначально все вопросы имели большие единицы кратности и обучаемому удалось дать ряд правильных ответов на один из вопросов, понизив его кратность до единицы, причем этого ему удалось добиться на фоне плохих ответов на все остальные вопросы, то дальнейшее рассмотрение этого вопроса будет как бы отложено до тех пор, пока обучаемый не усвоит еще по крайней мере один вопрос, сведя и его кратность к единице. Алгоритм препятствует слишком быстрому изъятию из рассмотрения отдельных вопросов.

71. Найти число операций прибавления единицы к r при выполнении алгоритма для заданного натурального n

```
r := 0;
for i = 1 to n do
   for j = i + 1 to n do
      for k = i + j − 1 to n do r := r + 1 od
   od
od
```

(иначе: определить, чему равно r). Формальное построение и последующее столь же формальное вычисление суммы по аналогии с проделанными в примере 12.1 привело бы к неправильному ответу (причина указывалась в § 12 в замечании к примеру 12.1).

72. Исследовать зависимость от целого $n > 0$ общего числа арифметических операций, требуемых алгоритмом

```
l := 0;
for i = 1 to n do
   for j = 1 to ⌊log2 i⌋ do l := l + i + j od
od
```

73. Исследовать зависимость от $n > 0$ общего числа арифметических операций, требуемых алгоритмом

```
l := 0;
for i = 1 to ⌊√n⌋ do
   z := i;
   while z > 1 do l := l + 1; z := z/7 od
od
```

(значения z не обязательно целые).

Глава 4

Нижняя граница сложности алгоритмов некоторого класса. Оптимальные алгоритмы

§ 14. Понятие нижней границы сложности

Хорошо известно, что существуют алгоритмически неразрешимые задачи. Но если даже задача алгоритмически разрешима, то возможно, что сложность каждого из алгоритмов ее решения будет в определенном смысле достаточно высокой.

Пример 14.1. Рассмотрим задачу поиска наименьшего элемента с помощью сравнений в массиве длины n. Для того чтобы иметь основания отсеять элемент массива как заведомо не являющийся наименьшим, необходимо, чтобы этот элемент участвовал хотя бы в одном сравнении и оказался бóльшим. Мы должны отсеять $n-1$ элемент, а в каждом сравнении ровно один элемент оказывается бóльшим. Это означает, что потребуется не менее $n-1$ сравнения.

Введем основное понятие этой главы.

Определение 14.1. Пусть $\mathscr{A}$ — класс алгоритмов решения некоторой задачи. Пусть принято соглашение о том, в чем измеряются затраты алгоритмов и что считается размером входа, и пусть n — обозначение размера входа. Функция $f(n)$ называется *нижней границей* сложности принадлежащих $\mathscr{A}$ алгоритмов, если для сложности $T_A(n)$ любого $A \in \mathscr{A}$ мы имеем $T_A(n) \geqslant f(n)$ при всех допустимых значениях размера входа n. (Если используются несколько параметров $n_1, n_2, \ldots, n_k$ размера входа, то нижняя граница — это, соответственно, функция аргументов $n_1, n_2, \ldots, n_k$.)

Это определение имеет смысл как для сложности в худшем случае, так и для сложности в среднем. Далее в примерах этого и следующего параграфов мы рассматриваем сложность в худшем случае; к сложности в среднем мы вернемся в § 17. Всюду будет рассматриваться вре-

менная сложность. Мы можем сформулировать обнаруженный при рассмотрении примера 14.1 факт следующим образом.

Предложение 14.1. *Функция $f(n) = n - 1$ является нижней границей сложности алгоритмов поиска наименьшего элемента массива длины n с помощью сравнений.*

(Иными словами, не существует такого алгоритма выбора наименьшего элемента массива длины n с помощью сравнений, сложность которого хотя бы для одного значения n была меньше чем $n - 1$.)

В роли класса $\mathscr{A}$ здесь выступает класс алгоритмов поиска наименьшего элемента массива с помощью сравнений. Если бы помимо сравнений были допустимы какие-то еще операции, то, возможно, $n - 1$ уже не годилось бы в качестве нижней границы. Например, если было бы разрешено пользоваться операцией выбора наименьшего из нескольких элементов, то задачу можно было бы решить одной операцией.

Пример 14.2. Рассмотрим класс алгоритмов сортировки с помощью сравнений. Если алгоритм работает для массивов любой длины, то, разумеется, можно рассмотреть этот алгоритм применительно к массивам некоторой фиксированной длины n. Любая сортировка с помощью сравнений может быть для каждого конкретного n изображена бинарным деревом. В корне и внутренних вершинах находятся выполняемые сравнения, в листьях выписаны результаты сортировки. Априори в исходном массиве возможен любой порядок элементов, поэтому дерево будет иметь $n!$ листьев. Если взять, например, сортировку простыми вставками, то при $n = 2$ ее можно изобразить деревом, представленным на рис. 18а. При $n = 3$ дерево принимает вид, показанный на рис. 18б. Сложность в худшем случае для каждого n равна высоте соответствующего дерева (напомним, что высотой листа называется число ребер в том единственном пути, который ведет от корня дерева к этому листу; высотой дерева называется максимум высот его листьев). Если высота некоторого бинарного дерева равна h, то, очевидно, оно содержит не более 2^h листьев (максимальное возможное число листьев достигается в случае полного бинарного дерева, которое имеет ровно 2^h листьев). Поэтому если $T(n)$ — это временная сложность некоторой сортировки сравнениями, то должно выполняться неравенство

$$2^{T(n)} \geqslant n!,$$

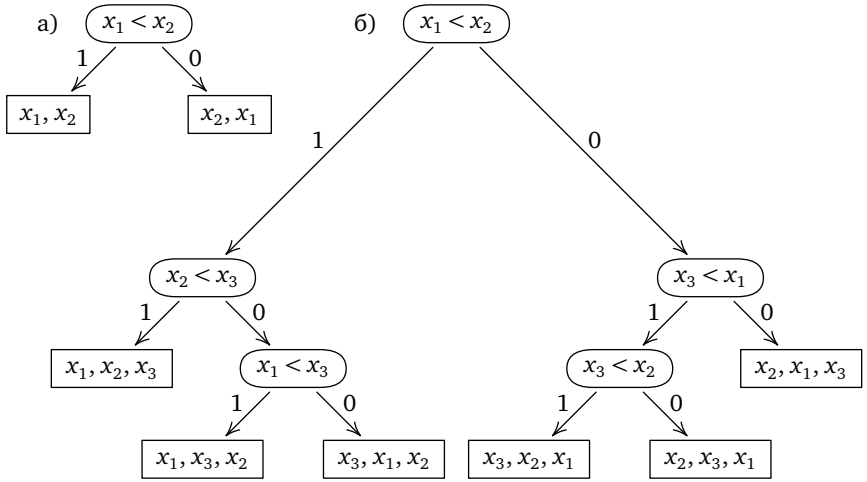


Рис. 18. Дерево сортировки простыми вставками для случаев а) $n = 2$ и б) $n = 3$.

откуда $T(n) \geqslant \log_2 n!$; так как значение $T(n)$ — целое число (мы измеряем сложность числом сравнений), то $T(n) \geqslant \lceil \log_2 n! \rceil$. Доказанное можно сформулировать в виде предложения.

Предложение 14.2. *Функция $f(n) = \lceil \log_2 n! \rceil$ является нижней границей сложности алгоритмов сортировки массивов длины n с помощью сравнений.*

Отсюда можно вывести, например, следующее предложение.

Предложение 14.3. *Пусть сложность $T(n)$ некоторой сортировки по числу сравнений не превосходит $n \log_2 n + cn$, где c — некоторая константа. Тогда $T(n) = n \log_2 n + O(n)$ и, как следствие, $T(n) \sim \sim n \log_2 n$.*

Доказательство. В силу предложения 14.2 и формулы (10.12) для всех достаточно больших n мы имеем $T(n) \geqslant \lceil \log_2 n! \rceil > n \log_2 n - 2n$, и, следовательно, $n \log_2 n - 2n < T(n) \leqslant n \log_2 n + cn$. Отсюда следует требуемое. □

Это позволяет, например, заключить, что для сложности по числу сравнений сортировки фон Неймана имеет место оценка $T_{\text{vN}}(n) = = n \log_2 n + O(n)$ и асимптотическое равенство $T_{\text{vN}}(n) \sim n \log_2 n$, — мы уже видели, что $T_{\text{vN}} < n\lceil \log_2 n \rceil < n \log_2 n + n$.

Добавим, что никакая сортировка не может иметь сложность $T(n)$ по числу сравнений, допускающую оценку $\alpha n \log_2 n + o(n \log n)$, $\alpha < 1$ (в частности, оценку $\alpha n \log_2 n + O(n)$), — это следует из того, что для всех достаточно больших n выполнено $T(n) > n \log_2 n - 2n$.

Пример 14.3. Обратимся к задаче поиска места элемента в упорядоченном массиве длины n.

Предложение 14.4. *Функция $f(n) = \lceil \log_2(n+1) \rceil$ является нижней границей сложности алгоритмов поиска места элемента в упорядоченном массиве длины n с помощью сравнений.*

Доказательство. Любой алгоритм поиска места элемента в упорядоченном массиве фиксированной длины n может быть изображен бинарным деревом. Число листьев такого дерева есть $n+1$. Далее рассуждаем так же, как при доказательстве предложения 14.2. □

Пример 14.4. Рассмотрим задачу вычисления a^n с помощью умножений (пример 4.2). Будем считать n размером входа. Затраты будем измерять количеством умножений.

Предложение 14.5. *Функция $f(n) = \lceil \log_2 n \rceil$ является нижней границей сложности алгоритмов вычисления a^n с помощью умножений.*

Доказательство. На каждом этапе алгоритма мы имеем вычисленный набор степеней

$$a^{m_1}, a^{m_2}, \ldots, a^{m_k}, \tag{14.1}$$

при этом на начальном этапе набор состоит из одного элемента a^1. Выполнив одно умножение, мы, очевидно, не можем увеличить максимальный показатель $m = \max\{m_1, m_2, \ldots, m_k\}$ более чем вдвое. Поэтому если определить на наборах вида (14.1) функцию L, равную $\log_2 n - \log_2 m$, то в результате одного шага (одного умножения) эта функция не может уменьшиться больше чем на 1. В то же время значение этой функции на исходном наборе равно $\log_2 n$, а на итоговом наборе она заведомо неположительна. □

§ 15. Оптимальные алгоритмы

Нижняя граница сложности класса алгоритмов определяется не единственным образом. Например, $f(n) = 0$ всегда является нижней границей, равно как и любая отрицательная функция. Чем больше найденная нижняя граница, тем она нетривиальнее и ценнее. Сигналом о том, что мы не сможем построить нижнюю границу бóльшую, чем

уже имеющаяся у нас нижняя граница $f(n)$, может служить, например, наличие $A \in \mathscr{A}$, для которого $T_A(n) = f(n)$. С такой ситуацией мы сталкиваемся в предыдущем параграфе в примерах 14.1 и 14.3. Известный нам алгоритм поиска наименьшего элемента и алгоритм бинарного поиска места элемента в упорядоченном массиве имеет каждый сложность, совпадающую с найденной нижней границей. Эти алгоритмы являются оптимальными в смысле следующего определения.

Определение 15.1. Пусть $\mathscr{A}$ — класс алгоритмов решения некоторой задачи. Пусть принято соглашение о том, в чем измеряются затраты алгоритмов и что считается размером входа, и пусть n — обозначение размера входа. Алгоритм $A \in \mathscr{A}$ называется *оптимальным* в $\mathscr{A}$, если $T_A(n)$ является нижней границей сложности алгоритмов из $\mathscr{A}$.

Пример 15.1. При получении нижней границы сложности и доказательстве оптимальности иногда бывает полезным привлечение функций на наборах тех величин, которые возникают в процессе выполнения алгоритма, например, на наборах значений переменных, используемых алгоритмом.

Предложение 15.1. *Функция* $f(n) = \left\lceil \frac{3n}{2} \right\rceil - 2$ *является нижней границей сложности алгоритмов одновременного выбора наибольшего и наименьшего элементов массива длины* n *с помощью сравнений.*

Доказательство. Каждый этап выполнения произвольного алгоритма V, основанного на сравнениях и предназначенного для поиска наибольшего и наименьшего элементов массива, может быть охарактеризован четверкой (A, B, C, D) подмножеств множества исходных элементов $\{x_1, x_2, \ldots, x_n\}$, где

- A состоит из всех тех элементов, которые вообще не сравнивались;
- B состоит из всех тех элементов, которые участвовали в некоторых сравнениях и всегда оказывались бóльшими;
- C состоит из всех тех элементов, которые участвовали в некоторых сравнениях и всегда оказывались меньшими;
- D состоит из всех тех элементов, которые участвовали в некоторых сравнениях, иногда оказываясь бóльшими, а иногда — меньшими.

Пусть a, b, c, d — количества элементов множеств A, B, C, D соответственно. Исходная ситуация характеризуется равенствами $a = n$, $b = c = d = 0$. После завершения алгоритма должны выполняться равен-

ства $a=0,\ b=c=1,\ d=n-2$. После первого сравнения на протяжении всего выполнения алгоритма постоянно будут иметь место неравенства $b\geqslant 1,\ c\geqslant 1$.

Все сравнения, совершаемые при выполнении алгоритма V, можно разбить на типы, обозначаемые $AA, AB, AC, AD,\ BB, BC, BD, CC, CD, DD$, например: сравнение принадлежит типу AB, если один из сравниваемых элементов берется из A, другой — из B, и т. д. Исходя из этого, можно выписать все возможные изменения четверки (a, b, c, d) под действием сравнений разных типов:

$$
\begin{aligned}
&AA\colon (a-2, b+1, c+1, d),\\
&AB\colon (a-1, b, c+1, d) \mid (a-1, b, c, d+1),\\
&AC\colon (a-1, b+1, c, d) \mid (a-1, b, c, d+1),\\
&AD\colon (a-1, b+1, c, d) \mid (a-1, b, c+1, d),\\
&BB\colon (a, b-1, c, d+1),\\
&BC\colon (a, b-1, c-1, d+2) \mid (a, b, c, d),\\
&BD\colon (a, b-1, c, d+1) \mid (a, b, c, d),\\
&CC\colon (a, b, c-1, d+1),\\
&CD\colon (a, b, c-1, d+1) \mid (a, b, c, d),\\
&DD\colon (a, b, c, d)
\end{aligned}
$$

(вертикальная черточка здесь означает «или»). Рассмотрим функцию $L(a, b, c, d)=\frac{3}{2}a+b+c-2$. Для начальной и конечной стадий имеем $L(n, 0, 0, 0)=\frac{3}{2}n-2$ и $L(0, 1, 1, n-2)=0$ соответственно.

Каждое сравнение типов AA, BB, CC понижает значение L на 1, т. е. дает $\Delta L=-1$. Выпишем все возможные значения ΔL при выполнении сравнений различных типов:

$$
\begin{aligned}
&AA, BB, CC\colon && -1,\\
&BD, CD\colon && -1 \mid 0,\\
&AB, AC\colon && -\frac{3}{2} \mid -\frac{1}{2},\\
&BC\colon && -2 \mid 0,\\
&AD\colon && -\frac{1}{2},\\
&DD\colon && 0.
\end{aligned}
$$

Сравнения, относящиеся к типам AB, AC, BC, могут приводить к $\Delta L<-1$, но это не может быть гарантировано никаким специальным выбором элементов из соответствующих множеств четверки

(A, B, C, D), даже если принимать во внимание результаты всех сравнений, в которые были вовлечены конкретные элементы этих множеств. Например, сравнение типа AB в том случае, когда выбранный элемент из A оказывается больше выбранного элемента из B, преобразует (a, b, c, d) в $(a-1, b, c, d+1)$, что дает $\Delta L < -1$. Но результаты предшествующих сравнений не дают оснований для отметания возможности того, что выбранный элемент из B равен $\max\{x_1, x_2, \ldots, x_n\}$ (ибо этот элемент оказался бóльшим во всех сравнениях, в которые он был вовлечен). Но тогда будет выполнено $\Delta L = -\frac{1}{2}$. Аналогичным образом дело обстоит для сравнений, принадлежащих типам AC, BC. Поэтому, рассматривая поведение алгоритма V в худшем случае, надо считать, что на всех этапах $\Delta L \geqslant -1$. Но тогда для достижения равенства $L(a, b, c, d) = 0$ потребуется не менее $\lceil L(n, 0, 0, 0) \rceil$ сравнений. Это значит, что общее число сравнений в худшем случае не меньше, чем $\left\lceil \frac{3n}{2} \right\rceil - 2$. □

Представленный в приложении А алгоритм одновременного поиска наибольшего и наименьшего элементов массива, требующий в худшем случае не более $\left\lceil \frac{3n}{2} \right\rceil - 2$ *сравнений, является оптимальным*[1].

Наряду с алгоритмами поиска наименьшего элемента, бинарного поиска места элемента в упорядоченном массиве и алгоритма одновременного поиска наибольшего и наименьшего элементов массива примером оптимального алгоритма может служить алгоритм («схема») Горнера вычисления значения полинома в данной точке (см. приложение D).

Для произвольного класса $\mathscr{A}$ оптимальный алгоритм может и не существовать: если, например, $\mathscr{A}$ содержит ровно два алгоритма, A_1, A_2, и при этом A_1 имеет низкую сложность при четных значениях целочисленного размера входа n и высокую — при нечетных, а A_2 — наоборот, то, очевидно, ни A_1, ни A_2 не будут оптимальными в $\mathscr{A}$. Ясно, что если оптимальные алгоритмы в классе $\mathscr{A}$ существуют, то их сложности совпадают: из неравенств $T_{A_1}(n) \geqslant T_{A_2}(n)$ и $T_{A_2}(n) \geqslant T_{A_1}(n)$ следует, что $T_{A_1}(n) = T_{A_2}(n)$. Несколько более содержательный пример можно найти в задаче 76.

В отличие от поиска наименьшего элемента, поиска места в упорядоченном массиве и одновременного поиска наибольшего и наи-

[1] Этот алгоритм и идея использования четверок (A, B, C, D) в доказательстве его оптимальности были предложены И. Полом в [56], но при этом убывающие функции в его доказательстве не привлекались, из-за чего потребовались дополнительные словесные мотивации.

меньшего элементов, задача сортировки не столь проста: те алгоритмы, которыми пользуются на практике для ее решения, не являются, строго говоря, оптимальными (см. приложение F).

Заметим при этом, что для каждого конкретного n за конечное число шагов может быть найдено наименьшее число сравнений, достаточное в худшем случае для сортировки n элементов, а вместе с ним и алгоритм сортировки с такой сложностью. Это следует из того, что при каждом конкретном n алгоритм сортировки может быть представлен бинарным деревом. Можно порождать одно за другим все бинарные деревья с высотой, не превосходящей $\lceil \log_2 n! \rceil$, в вершинах каждого такого дерева различными способами расставлять выражения вида $x_i < x_j$, где $i, j \leqslant n$, а в листьях — различные перестановки длины n. Для каждого размеченного таким способом дерева нужно проверить, действительно ли каждая ветвь приводит именно к тому порядку, который указан в листе, и что все возможные порядки охвачены. Из всех «правильных» деревьев выбирается имеющее наименьшую высоту. Оно определяет оптимальный алгоритм для заданного n.

Таким образом, мы имеем алгоритм, который, исходя из $n > 0$, строит оптимальный по числу сравнений алгоритм сортировки массивов из n элементов (алгоритм строит алгоритм). Этот алгоритм построения оптимального алгоритма сортировки требует огромной работы, даже если применить все средства экономии перебора. Этот пример еще раз показывает, что понятие алгебраической сложности требует осторожного обращения, — объем неучитываемых операций может превзойти разумные пределы.

Бинарный алгоритм возведения в степень n также не является оптимальным по числу умножений при использовании n в качестве размера входа (см. задачу 19), хотя и легко показать, что для случая $n = 2^k$ при рассмотрении k в качестве размера входа оптимальность имеет место: из предложения 14.5 следует, что возведение в степень $n = 2^k$ не может быть выполнено с затратами меньшими, чем k умножений, а бинарный алгоритм обходится в точности этим числом умножений.

В общем же случае, как и многие известные алгоритмы сортировки, бинарный алгоритм возведения в степень оптимален лишь в некотором асимптотическом смысле, о чем пойдет речь в следующем параграфе.

Если затраты для каждого входа являются целыми числами (например, они являются количеством выполненных операций), то для фиксированного размера n_0 входа в рассматриваемом классе $\mathscr{A}$ ал-

горитмов решения некоторой задачи P существует такой, сложность которого при $n = n_0$ есть минимум сложностей всех алгоритмов из $\mathscr{A}$. Можно определить такой минимум для любого значения n, получаемую таким способом функцию от n иногда называют *сложностью задачи P* по отношению к классу алгоритмов $\mathscr{A}$. Важно, что при разных n минимумы могут доставляться разными алгоритмами из $\mathscr{A}$ (см. задачу 76). В дальнейшем мы не будем касаться этих вопросов.

§ 16. Асимптотические нижние границы. Алгоритм, оптимальный по порядку сложности

Известно не очень много алгоритмов, оптимальных в смысле определения 15.1. Рассмотрим дополнительно понятие алгоритма, оптимального по порядку сложности.

Определение 16.1. Пусть $\mathscr{A}$ — класс алгоритмов решения некоторой задачи. Пусть принято соглашение о том, в чем измеряются затраты алгоритмов и что считается размером входа, и пусть n — обозначение размера входа. Функция $f(n)$ называется *асимптотической нижней границей* сложности принадлежащих $\mathscr{A}$ алгоритмов, если для сложности $T_A(n)$ любого $A \in \mathscr{A}$ мы имеем $T_A(n) = \Omega(f(n))$. (Если используются несколько параметров $n_1, n_2, \ldots, n_k$ размера входа, то асимптотическая нижняя граница — это, соответственно, функция аргументов $n_1, n_2, \ldots, n_k$.) Алгоритм $A \in \mathscr{A}$ называется *оптимальным по порядку сложности* в $\mathscr{A}$, если $T_A(n)$ является асимптотической нижней границей сложности алгоритмов из $\mathscr{A}$.

Очевидно, что любая неотрицательная нижняя граница сложности является и асимптотической нижней границей, и что любой оптимальный алгоритм является оптимальным по порядку сложности.

Ниже в примере 16.4 мы увидим, что может существовать несколько оптимальных по порядку сложности в $\mathscr{A}$ алгоритмов. Однако сложности таких алгоритмов оказываются величинами одного порядка.

Теорема 16.1. *Пусть $A, B \in \mathscr{A}$ оптимальны по порядку сложности в $\mathscr{A}$. Тогда $T_A(n) = \Theta(T_B(n))$.*

Доказательство. Так как алгоритм A оптимален по порядку сложности, имеем $T_B(n) = \Omega(T_A(n))$ или, что то же самое, $T_A(n) = O(T_B(n))$, что вместе с $T_A(n) = \Omega(T_B(n))$ (алгоритм B оптимален по порядку сложности) дает $T_A(n) = \Theta(T_B(n))$. □

Укажем достаточное условие оптимальности алгоритма по порядку сложности.

Теорема 16.2. *Если функция $f(n)$ является асимптотической нижней границей сложности принадлежащих классу $\mathscr{A}$ алгоритмов и если алгоритм $A \in \mathscr{A}$ таков, что $T_A(n) = O(f(n))$, то этот алгоритм оптимален в $\mathscr{A}$ по порядку сложности и $T_A(n) = \Theta(f(n))$.*

Доказательство. Для произвольного $B \in \mathscr{A}$ имеем $T_B(n) = \Omega(f(n))$; учитывая, что $T_A(n) = O(f(n))$, получаем $T_B(n) = \Omega(T_A(n))$, т. е. алгоритм A оптимален по порядку сложности.

Из $T_A(n) = \Omega(f(n))$ и $T_A(n) = O(f(n))$ получаем $T_A(n) = \Theta(f(n))$. □

Пример 16.1. Из предложения 14.5 следует, что функция $f(n) = \log_2 n$ является асимптотической нижней границей сложности алгоритмов вычисления a^n с помощью умножений (эта функция даже является нижней границей в смысле определения 14.1). Для сложности бинарного алгоритма вычисления a^n мы имеем $T_{\mathrm{RS}}(n) = O(\log n)$ (см. пример 4.2). Из теоремы 16.2 теперь получаем:

Бинарный алгоритм возведения в степень является оптимальным по порядку сложности в классе алгоритмов вычисления a^n с помощью умножений.

Иногда, хотя не часто, из совершенно элементарных соображений удается найти такую нижнюю границу сложности, что далее по теореме 16.2 оказывается возможным обосновать оптимальность некоторого известного алгоритма по порядку сложности. Рассмотрим два примера из теории графов.

Пример 16.2. Вопрос о существовании и построении эйлерова цикла данного ориентированного графа сформулирован в задаче 16. Остановимся на случае связного графа. Очевидно, что если избрать число ребер $|E|$ в качестве размера входа, то $|E|$ является нижней границей сложности класса всех алгоритмов построения эйлеровых циклов, так как, во-первых, любой эйлеров цикл содержит по определению все ребра графа и на любое ребро уйдет по крайней мере одна операция, и, во-вторых, для любого натурального n существует граф с $|E| = n$, содержащий эйлеров цикл. Поэтому тот алгоритм со сложностью $O(|E|)$, который требуется дать в задаче 16, будет оптимальным по порядку сложности.

Пример 16.3. Рассмотрим задачу построения минимального остовного (покрывающего) дерева данного связного графа, каждое ребро

которого имеет некоторый неотрицательный вес. Имеется в виду дерево:

(a) которое охватывает все вершины данного графа,
(b) все ребра которого являются ребрами данного графа,
(c) которое среди всех деревьев, обладающих свойствами (a), (b), имеет минимальный суммарный вес ребер [1].

Будем считать, что граф не содержит кратных ребер. Пример графа с соответствующим остовным деревом дан на рис. 19. Для построения минимального остовного дерева известны остроумные алгоритмы, например, алгоритм Р. Прима [2] с оценкой сложности

$$O(|E|+|V|\log|V|). \qquad (16.1)$$

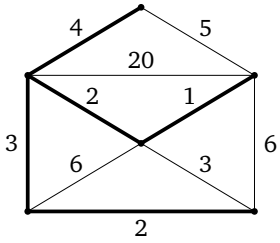


Рис. 19. Граф с приписанными ребрам весами и его остовное дерево.

В этой оценке $|V|$ и $|E|$ рассматриваются как два параметра размера входа. Не обсуждая вопрос, является или нет алгоритм Прима оптимальным по порядку сложности при таком выборе параметров размера входа и не рассматривая детали алгоритма Прима, мы покажем, что этот алгоритм оптимален по порядку сложности при использовании $|V|$ как размера (единственного параметра размера) входа. Так как среди всех графов без кратных ребер, имеющих $|V|$ вершин, наибольшее число ребер

$$\frac{|V|(|V|-1)}{2}$$

имеет полный граф, то оценка (16.1) имеет следствием оценку $O(|V|^2)$. С другой стороны, асимптотическая нижняя граница $\Omega(|V|^2)$ для алгоритмов такого рода получается тривиально: эту оценку допускает число ребер полного графа, а на каждое ребро графа алгоритм затрачивает по крайней мере одну операцию (он не может не принять в расчет вес какого-либо ребра).

Если использовать для входов алгоритмов построения остовного дерева размер $|V|$, то $|V|^2$ даст асимптотическую нижнюю границу сложности этих алгоритмов, и алгоритм Прима будет оптимальным по порядку сложности (*его сложность, соответствующая этому размеру входа, есть* $\Theta(|V|^2)$).

[1] См. [21, гл. 24].
[2] Этот алгоритм описан, например, в [21, разд. 24.2].

Из этого примера видно, что само свойство оптимальности находится в зависимости от выбора размера входа [1].

В следующем примере, касающемся сортировки массивов, рассматривается сложность в худшем случае по числу сравнений.

Пример 16.4. Аналогично тому, как это сделано в примере 16.1, мы получаем, что любая сортировка, сложность которой допускает оценку $O(\log n!)$, является оптимальной по порядку сложности. В силу формулы Стирлинга мы имеем $n \log_2 n = O(\log n!)$, откуда следует, что любая сортировка, сложность которой по числу сравнений допускает оценку $O(n \log n)$, является оптимальной по порядку сложности.

Сортировка бинарными вставками и сортировка фон Неймана являются оптимальными по порядку сложности по числу сравнений.

(Позднее мы установим оценку $O(n \log n)$ и для сложности рекурсивной сортировки слияниями.) Эти сортировки имеют разные сложности как функции от n: например, $T_{\mathrm{vN}}(5) = 9$, $T_{\mathrm{B}}(5) = 8$. Но, повторим, их сложности являются величинами одного порядка при $n \to \infty$.

Резюмируем наши наблюдения.

Предложение 16.1. *Необходимым и достаточным условием оптимальности сортировки по порядку сложности является справедливость оценки $O(n \log n)$ для сложности этой сортировки. Если для некоторой сортировки справедлива оценка $O(n \log n)$, то справедлива и оценка $\Theta(n \log n)$.*

Доказательство. Достаточность уже установлена выше. Оставшаяся часть доказательства следует, например, из теоремы 16.1 и оптимальности по порядку сложности сортировки бинарными вставками. □

Заметим при этом, что для сортировок бинарными вставками и фон Неймана справедливо значительно более сильное утверждение, чем утверждение об их оптимальности по порядку сложности. Для их сложностей и сложности $T_{\mathrm{opt}}(n)$ оптимального алгоритма имеет место асимптотическая эквивалентность:

$$T_{\mathrm{B}}(n) \sim T_{\mathrm{vN}}(n) \sim T_{\mathrm{opt}}(n) \sim n \log_2 n \sim \log_2 n!.$$

Коснемся нижних границ пространственной сложности алгоритмов сортировки. Если из всех этих алгоритмов выделить класс таких, которые используют для перемещения элементов обмены ($\leftrightarrow$), а не присва-

[1] Этот пример сообщил автору Е. В. Зима.

ивания (:=), то для этого класса единственной нижней границей, если говорить о функциях, принимающих неотрицательные значения, будет тождественно равная нулю функция, так как существуют, например, пузырьковая сортировка, сортировки простыми и бинарными вставками, не использующие дополнительных переменных того типа, к которому относятся элементы сортируемого массива. (То, что требуются переменные для хранения значений индексов элементов, несущественно при изучении алгебраической сложности сортировки.) Алгоритмы, использующие присваивания для перемещения сортируемых элементов, не могут, естественно, обойтись без дополнительного места для хранения хотя бы одной величины того же типа, что и элементы сортируемого массива, и одной из нижних границ будет функция, тождественно равная единице. Если рассматривать все алгоритмы сортировки вместе, то вновь единственная неотрицательная целочисленная граница — это 0. В соответствии с этим определяются как оптимальные, так и оптимальные по порядку сложности алгоритмы.

Для произвольного класса $\mathscr{A}$ оптимальный по порядку сложности алгоритм может и не существовать: если $\mathscr{A}$ содержит всего два алгоритма A_1, A_2 и при этом A_1 имеет низкую сложность при четных значениях целочисленного размера входа n и высокую — при нечетных, а A_2 — наоборот, то, очевидно, ни A_1, ни A_2 не будут оптимальными в $\mathscr{A}$ по порядку сложности (представим себе, что $T_{A_1} = (1 + (-1)^n)n$, $T_{A_2} = (1 - (-1)^n)n$). В то же время, если допустить, что $T_{A_1} = (2 + (-1)^n)n$, $T_{A_2} = (2 - (-1)^n)n$, то $T_{A_1}(n) = \Theta(T_{A_2}(n))$ (обе функции имеют порядок n), и это говорит о возможности существования оптимального по порядку сложности алгоритма при отсутствии оптимального.

§ 17. Нижняя граница сложности в среднем

При рассмотрении сложности в среднем мы основываемся на тех же понятиях нижней границы, оптимального и оптимального по порядку сложности алгоритма, которые были введены для сложности в худшем случае.

Пример 17.1. Вновь обратимся к классу алгоритмов сортировки. Будем, как и в § 6, рассматривать вероятностное пространство Π_n.

Предложение 17.1. *Функция* $\log_2 n!$ *является нижней границей сложности в среднем для класса алгоритмов сортировки массивов длины* n *с помощью сравнений.*

Прежде всего докажем вспомогательное утверждение.

Лемма 17.1. *В любом двоичном дереве с m листьями сумма высот всех листьев не меньше $m \log_2 m$.*

Доказательство. Пусть непусто множество M всех двоичных деревьев, для которых сумма высот всех листьев меньше $m \log_2 m$. Выберем в M какое-нибудь из деревьев, имеющих наименьшую сумму высот, и обозначим его U (сумму высот всех листьев будем обозначать через $H(U)$). Это дерево не может состоять из одного лишь корня, потому что для такого дерева обсуждаемое неравенство выполнено. Далее, из корня этого дерева не может выходить только одно ребро, потому что для дерева, получающегося из исходного удалением корня и этого ребра, обсуждаемое неравенство не выполняется, и такое новое дерево имеет сумму высот меньшую, чем дерево U (противоречие со способом выбора дерева U). Поэтому из корня дерева U выходит два ребра, концы которых являются корнями двух деревьев U_1 и U_2, для которых выполнено обсуждаемое неравенство. Пусть эти деревья имеют, соответственно, m_1 и m_2 листьев, $m_1, m_2 > 0$. Имеем соотношение для сумм высот всех листьев

$$H(U) = H(U_1) + H(U_2) + m \geqslant m_1 \log_2 m_1 + m_2 \log_2 m_2 + m.$$

Отсюда получаем

$$H(U) \geqslant m_1 \log_2 m_1 + (m - m_1) \log_2 (m - m_1) + m$$

при некотором m_1 таком, что $1 \leqslant m_1 \leqslant m - 1$.

Легко показать, что при фиксированном m функция $f(x) = x \log_2 x + (m-x) \log_2 (m-x)$ достигает минимума на отрезке $[1, m-1]$ в точке (не обязательно целой) $m/2$. Поэтому

$$H(U) \geqslant \frac{m}{2} \log_2 \frac{m}{2} + \frac{m}{2} \log_2 \frac{m}{2} + m = m \log_2 m.$$

Но это противоречит способу выбора U. Лемма доказана. □

Доказательство предложения 17.1. Рассмотрим какое-либо дерево сортировки массива длины n. Это дерево имеет ровно $n!$ листьев, каждый лист соответствует перестановке элементов исходного массива. Появление на выходе рассматриваемой сортировки любой из этих перестановок имеет одну и ту же вероятность $\frac{1}{n!}$, количество сравнений, приводящее к этой перестановке — это высота соответствующего этой перестановке листа. Поэтому математическое ожидание числа сравнений равно произведению суммы высот всех листьев на $\frac{1}{n!}$.

Согласно лемме 17.1 сумма высот всех листьев должна быть не меньше, чем $n!\log_2 n!$. Отсюда математическое ожидание числа сравнений не меньше, чем $\log_2 n!$. □

Доказанные ранее теорема 16.2 и предложение 16.1 справедливы в равной мере и для сложности в худшем случае, и для сложности в среднем. Принимая это во внимание, мы получаем следствие предложения 17.1:

Сортировка бинарными вставками и сортировка фон Неймана, а также быстрая сортировка являются оптимальными по порядку сложности в среднем по числу сравнений.

К этому списку позднее мы добавим и рекурсивную сортировку слияниями. Наиболее же существенно упоминание в этом списке быстрой сортировки. С одной стороны, как мы знали и раньше, эта сортировка очень удобна и имеет низкую пространственную сложность, с другой — мы видим теперь, что и в смысле временной сложности в среднем эта сортировка в определенном смысле может быть отнесена к наилучшим.

Можно показать существование оптимальной в среднем сортировки: для каждого фиксированного n в множестве всех деревьев сортировки массивов длины n можно рассмотреть подмножество деревьев, имеющих наименьшую сумму H высот всех листьев (тогда и $H/n!$ будет иметь наименьшее значение), и взять какое-нибудь из деревьев этого подмножества. Определяя сортировку этим способом для всех n мы получаем оптимальную сортировку. Для любой оптимальной в среднем сортировки выполнено

$$\overline{T}_{\text{opt}}(n) \sim \log_2 n! \sim n\log_2 n,$$

так как, например, мы имеем $T_{\text{vN}}(n) \sim \log_2 n!$ и

$$T_{\text{vN}}(n) \geqslant \overline{T}_{\text{vN}}(n) \geqslant \overline{T}_{\text{opt}}(n) \geqslant \log_2 n!.$$

В этом примере, как и во всем этом параграфе, мы не касаемся рандомизированных алгоритмов, о которых будет говориться в § 18.

Как мы ранее наблюдали в некоторых примерах, рассмотрение функции L, подобранной надлежащим образом, и изучение изменения ее значений в ходе выполнения алгоритма нередко позволяет получать хорошие оценки (в частности, нижние границы) сложности в худшем случае. Рассмотрение такого рода функций может приводить к цели и при исследовании сложности в среднем. В задаче 37 функцию L можно определить как максимум значений компонент

инверсионного вектора перестановки. Эта функция является случайной величиной на Π_n, при этом $\Delta L = -1$ на каждом шаге алгоритма. В других ситуациях начальное значение L может определяться однозначно, тогда как ΔL является случайной величиной. Сформулируем теорему, полезную в этих ситуациях.

Теорема 17.1. *Пусть $\xi_1, \xi_2, \ldots$ — последовательность неотрицательных случайных величин на некотором вероятностном пространстве. Пусть числовая последовательность $h_1, h_2, \ldots$ такова, что для каждого $k \geqslant 1$ выполнено $\mathsf{E}(\xi_k | \xi_1, \xi_2, \ldots, \xi_{k-1}) \leqslant h_k$ при всех значениях $\xi_1, \xi_2, \ldots, \xi_{k-1}$. Зафиксируем неотрицательное число q и введем целочисленную случайную величину*

$$\tau = \min\left\{ n : \sum_{k=1}^{n} \xi_k \geqslant q \right\}.$$

Пусть $\tau < \infty$ всюду на рассматриваемом вероятностном пространстве. Тогда $\mathsf{E}\left(\sum_{k=1}^{\tau} h_k \right) \geqslant q$.

Доказательство приведено в приложении Е.

Для того чтобы воспользоваться этой теоремой, можно опять попытаться определить некоторую неотрицательную функцию L, отражающую степень «недорешенности» рассматриваемой задачи: значение L равно нулю, если алгоритм доработал до конца и задача решена. Считаем, что всем входам фиксированного размера сопоставляется одно и то же значение функции L. После выбора такой функции L мы должны показать, что последовательность величин $\mathsf{E}(-\Delta L)$, соответствующих идущим друг за другом этапам алгоритма, ограничена сверху некоторой известной числовой последовательностью $h_1, h_2, \ldots$; тогда случайная величина τ, равная для данного входа общему количеству этапов, приводящих к завершению алгоритма, такова, что $\mathsf{E}\left(\sum_{k=1}^{\tau} h_k \right) \geqslant q$, где q — значение функции L, сопоставляемое входам алгоритма рассматриваемого фиксированного размера. Это неравенство можно использовать, чтобы оценить снизу значение $\mathsf{E}\tau$. Конечность величины τ следует из завершимости алгоритма для любого входа.

Пример 17.2. Вернемся к задаче одновременного выбора наибольшего и наименьшего элементов массива длины n, $n \geqslant 2$, с помощью сравнений (пример 15.1). Вероятностным пространством здесь вновь будем считать Π_n. Каждая перестановка из Π_n отражает, как обычно,

взаимный порядок элементов исходного массива. Все перестановки считаются равновероятными.

Чтобы воспользоваться теоремой 17.1, мы полагаем, что случайные величины $\xi_1, \xi_2, \ldots$ для произвольного алгоритма одновременного выбора наибольшего и наименьшего элементов равны значениям $-\Delta L$ на последовательных шагах этого алгоритма, об определении функции L будет сказано ниже. Значения $\xi_1, \xi_2, \ldots$ зависят от конкретной перестановки из Π_n, отражающей взаимный порядок элементов в исходном массиве. После того, как алгоритм закончил работу, значения ξ_k при последующих значениях k равны нулю.

Предложение 17.2. *Функция*

$$f(n) = \begin{cases} \frac{3}{2}n - 2, & \text{если } n \text{ четно}, \\ \frac{3}{2}n - 2 + \frac{1}{2n}, & \text{если } n \text{ нечетно}, \end{cases} \tag{17.1}$$

является нижней границей сложности в среднем алгоритмов одновременного выбора наибольшего и наименьшего элементов массива длины n, $n \geqslant 2$, с помощью сравнений.

Доказательство. Вновь обратимся к множествам A, B, C, D, к рассмотренным в доказательстве предложения 15.1 типам $AA, AB, \ldots, DD$ сравнений, количествам a, b, c, d элементов множеств A, B, C, D и функции $L(a, b, c, d) = \frac{3}{2}a + b + c - 2$ (равенство $L = 0$ является необходимым и достаточным условием того, что искомые элементы найдены).

Пусть в процессе выбора наибольшего и наименьшего элементов массива уже произведены некоторые сравнения элементов этого массива. При каждом из этих сравнений получался некоторый результат «истина» или «ложь», и эти результаты нам известны. Если процесс выбора наибольшего и наименьшего элементов еще не завершен, то следующим шагом вновь будет некоторое сравнение одного из типов $AA, AB, \ldots, DD$. Сравнения AB, AC и BC представляют особый интерес, так как можно бы предположить, что здесь получится $\mathsf{E}\Delta L < -1$. Но на самом деле это неравенство не имеет места ни при одном из этих трех типов сравнений.

Тип BC. Покажем невозможность такого выбора индексов i и j, основанного только на результатах уже произведенных сравнений, что $x_i \in B$, $x_j \in C$ и при этом с вероятностью, не меньшей половины, выполняется $x_i < x_j$.

Пусть сделан некий выбор индексов i и j, при котором $x_i \in B$, $x_j \in C$. Рассмотрим множество M всех массивов $y_1, y_2, \ldots, y_n$, которые

получаются перестановками элементов массива $x_1, x_2, \ldots, x_n$ и при этом сравнения элементов с теми же самыми индексами, которые использовались в уже произведенных сравнениях элементов массива $x_1, x_2, \ldots, x_n$, дают для массивов из M результаты, совпадающие с полученными для $x_1, x_2, \ldots, x_n$. Представим множество M в виде объединения двух непересекающихся подмножеств M_1 и M_2, где в M_1 входят те массивы, для которых $y_i < y_j$, а в M_2 — те, для которых $y_j < y_i$. Покажем, что в M_1 больше элементов, чем в M_2. Операция обмена $y_i \leftrightarrow y_j$ определяет взаимно однозначное отображение на себя множества всех массивов, которые получаются перестановками элементов массива $x_1, x_2, \ldots, x_n$. Обратным к этому отображению является оно само. В силу определения множеств B и C это отображение переводит элементы множества M_1 в элементы множества M_2. Следовательно, в M_1 элементов не больше, чем в M_2. Но в M_2 найдется массив, который при этом отображении не переходит в массив из M_1 и, более того, вообще покидает множество M (чтобы убедиться в этом, достаточно заметить, что в M_2 имеется такой массив $y_1, y_2, \ldots, y_n$, для которого $y_j = \min\{y_1, y_2, \ldots, y_n\}$). Поэтому в M_1 элементов меньше, чем в M_2. Если m_1, m_2 обозначают количества элементов множеств M_1 и M_2, то имеем

$$\frac{m_1}{m_1 + m_2} < \frac{m_2}{m_1 + m_2}$$

и, следовательно,

$$\frac{m_1}{m_1 + m_2} < \frac{1}{2}.$$

Таким образом, вероятность того, что i-й элемент меньше j-го, есть $\frac{1}{2} - \varepsilon$, $\varepsilon > 0$. Получаем

$$\mathsf{E}\Delta L = -2\left(\frac{1}{2} - \varepsilon\right) = -1 + 2\varepsilon > -1.$$

Тип AB. Аналогично предыдущему можно показать, что при любом способе выбора индексов таких, что $x_i \in A$, $x_j \in B$, вероятность выполнения неравенства $x_i > x_j$ есть $\frac{1}{2} - \varepsilon$, $\varepsilon > 0$. Получаем

$$\mathsf{E}\Delta L = -\frac{3}{2}\left(\frac{1}{2} - \varepsilon\right) - \frac{1}{2}\left(\frac{1}{2} + \varepsilon\right) = -1 + \varepsilon > -1.$$

Тип AC. Рассматривается аналогично типу *AB*.

В дальнейшем будет полезна оценка ε для сравнений типов *AB* и *AC*. Рассмотрим для определенности *AB*. Очевидно, что чем больше сравнений к рассматриваемому моменту прошел элемент $x_j \in B$, тем меньше вероятность того, что элемент $x_i \in A$ окажется больше него.

Наибольшая же вероятность получится в случае, когда x_j сравнивался только с одним элементом, например с x_s, и x_s — это наименьший элемент исходного массива. Вычислим вероятность, с которой в этом случае $x_i > x_j$. Итак, если упорядочить все элементы, кроме x_i, x_j, то x_s окажется наименьшим (первым). Общее число способов, которыми к этой упорядоченной совокупности можно добавить x_i, x_j, с учетом $x_s < x_j$ равно $n(n-2)$ (для добавления x_j существует $n-2$ способа, затем добавляем x_i, и для этого существует n способов). Из них $\binom{n}{2} - 1$ соответствуют неравенству $x_i < x_j$ (возможен любой выбор двух среди n мест, кроме двух первых). Интересующая нас вероятность равна

$$\frac{n(n-2) - \binom{n}{2} + 1}{n(n-2)} = \frac{1}{2} - \frac{1}{2n}.$$

Это дает нам $\varepsilon \geqslant \frac{1}{2n}$, и, таким образом,

$$\mathsf{E}\Delta L \geqslant -1 + \frac{1}{2n}$$

для AB и AC. Если при четном n удастся обойтись сравнениями типов AA, BB, CC, то сравнений потребуется в точности $\frac{3}{2}n - 2$. Если n нечетно, то для решения задачи обязательно потребуется произвести хотя бы одно сравнение типа AB, AC или AD. Сравнение типа AD дает $\Delta L = -\frac{1}{2} > -1 + \frac{1}{2n}$.

Можно убедиться и в том, что для сравнений типов AB и AC выполняется $\mathsf{E}\Delta L \geqslant -1 + \frac{1}{2n}$, если $\mathsf{E}\Delta L$ рассматривается как математическое ожидание при условии, что значения ΔL для всех предыдущих сравнений, предписываемых алгоритмом, известны. Эти известные значения определяют некоторое событие V в вероятностном пространстве Π_n. Событие V может быть представлено как сумма $V_1 + V_2 + \ldots + V_l$ попарно несовместных событий, где каждое V_k состоит из тех элементов Π_n, для которых все предыдущие сравнения образуют некоторую фиксированную последовательность сравнений элементов с вполне определенными для данного k индексами, и при этом возникающие значения ΔL совпадают с известными из условия. В силу доказанного выше $\mathsf{E}(\Delta L | V_k) \geqslant -1 + \frac{1}{2n}$ для каждого $k \leqslant l$. Отсюда по формуле полного математического ожидания $\mathsf{E}(\Delta L | V) \geqslant -1 + \frac{1}{2n}$.

В сумме $\sum_{k=1}^{\tau} h_k$, о которой идет речь в теореме 17.1, мы можем положить $h_k = 1$ для всех значений k, кроме какого-то одного значения k_0,

для которого $h_{k_0} = 1 - \frac{1}{2n}$. Это дает нам

$$\mathsf{E}\Big(\sum_{k=1}^{\tau} h_k\Big) = \mathsf{E}(\tau - 1) + 1 - \frac{1}{2n},$$

и с помощью следующего из теоремы 17.1 соотношения

$$\mathsf{E}\Big(\sum_{k=1}^{\tau} h_k\Big) \geqslant \frac{3}{2}n - 2$$

мы находим

$$\mathsf{E}(\tau - 1) + 1 - \frac{1}{2n} \geqslant \frac{3}{2}n - 2.$$

После упрощения получаем

$$\mathsf{E}\tau \geqslant \frac{3}{2}n - 2 + \frac{1}{2n}.$$

Таким образом, все доказано и для четных, и для нечетных n. □

Если существует алгоритм решения рассматриваемой задачи, который требует ровно (17.1) сравнений, то, по только что доказанному предложению, этот алгоритм является оптимальным в среднем по числу сравнений. Укажем такой алгоритм. Если n четно, то действуем в соответствии с алгоритмом, обсуждавшимся в примере 15.1. Пусть $n = 2k + 1,\ k \geqslant 0$. Находим

$$m_i = \min\{x_{2i-1}, x_{2i}\}, \quad M_i = \max\{x_{2i-1}, x_{2i}\},$$

$i = 1, 2, \ldots, k$, и

$$m = \min\{m_1, m_2, \ldots, m_k\},$$

что потребует $n - 2$ сравнений. Без дополнительных затрат можно найти s, $1 \leqslant s \leqslant 2k$, такое, что $m = m_s$. Далее сравниваем x_n (т. е. x_{2k+1}) с M_s. Согласно рассуждениям, проведенным выше при анализе сравнений типа AB, вероятность того, что $x_n > M_s$, равна $\frac{1}{2} - \frac{1}{2n}$. Если $x_n > M_s$, то результатом работы алгоритма будет

$$\max\{M_1, \ldots, M_{s-1}, M_{s+1}, \ldots, M_k, x_n\}, \quad m,$$

в противном случае —

$$\max\{M_1, M_2, \ldots, M_k\}, \quad \min\{m, x_n\}.$$

Среднее число сравнений при нечетном n равно

$$(n-2) + \Big(\frac{1}{2} - \frac{1}{2n}\Big) + 2\Big(\frac{1}{2} + \frac{1}{2n}\Big) + \Big(\frac{n-1}{2} - 1\Big) = \frac{3}{2}n - 2 + \frac{1}{2n},$$

что и требовалось.

Существует оптимальный в среднем алгоритм одновременного нахождения наибольшего и наименьшего элементов массива длины n, $n \geqslant 2$, с помощью сравнений. Этот алгоритм имеет сложность (17.1).

§ 18. Нижние границы сложности рандомизированных алгоритмов. Принцип Яо

Этот параграф начнем с формулировки и обсуждения одного факта из теории игр. С произвольной квадратной числовой матрицей $M = (m_{ij})$ порядка k связывается *матричная игра* двух игроков, которых мы назовем I и J. Один раунд игры состоит в том, что I и J независимо друг от друга называют соответственно целые i и j из диапазона от 1 до k, и после этого J выплачивает своему сопернику I сумму в m_{ij} рублей (при $m_{ij} < 0$ сумма в $-m_{ij}$ рублей выплачивается игроком I игроку J). *Стратегией* называется вектор $p = (p_1, p_2, \ldots, p_k)$ с вещественными неотрицательными компонентами, в сумме дающими единицу; стратегии $(1, 0, \ldots, 0)$, $(0, 1, \ldots, 0)$, $\ldots$, $(0, 0, \ldots, 1)$ называются *чистыми*. Допустим, что игроки независимо избирают для себя некоторые стратегии — обозначим их соответственно p и q, — и затем в последовательных раундах игрок I называет число i с вероятностью p_i, а игрок J называет число j с вероятностью q_j. Математическое ожидание выигрыша игрока I равно, как нетрудно подсчитать,

$$\sum_i \sum_j m_{ij} p_i q_j,$$

эту величину мы будем обозначать $M(p, q)$. Допустим также, что вместо проведения большого числа раундов игры каждый из игроков, основываясь на известной матрице M, один раз выбирает, независимо от другого игрока, для себя некоторую стратегию и сообщает ее судье, который по полученным от игроков стратегиям (векторам p и q) вычисляет $M(p, q)$, умножает его на количество предполагавшихся раундов и объявляет полученное число выигрышем игрока I. Естественно, что игрок I заинтересован в нахождении такой стратегии, которая максимизирует $M(p, q)$, а у игрока J цели прямо противоположные. Одна из теорем раздела теории игр, посвященного матричным играм[1], утверждает, что для игроков существуют *оптимальные* стратегии p^*, q^* такие, что $M(p^*, q^*)$ является одновременно значением величин

$$\max_p \min_q M(p, q) \quad \text{и} \quad \min_q \max_p M(p, q).$$

[1] См., например, [7, теорема 2.1].

В дальнейшем нас не будут интересовать оптимальные стратегии, однако равенство

$$\max_p \min_q M(p,q) = \min_q \max_p M(p,q) \tag{18.1}$$

окажется для нас очень ценным.

При фиксированной стратегии p значение $\min\limits_q M(p,q)$ равно минимальному значению среди

$$\sum_i m_{i1}p_i, \quad \sum_i m_{i2}p_i, \quad \ldots, \quad \sum_i m_{ik}p_i,$$

т. е. $\min\limits_j \sum\limits_i m_{ij}p_i$ (в качестве стратегии q, на которой достигается минимум, выступает некоторая чистая стратегия). Аналогичным образом $\max\limits_p M(p,q) = \max\limits_i \sum\limits_j m_{ij}q_j$. Это позволяет переписать равенство (18.1) в виде

$$\max_p \min_j \sum_i m_{ij}p_i = \min_q \max_i \sum_j m_{ij}q_j, \tag{18.2}$$

откуда следует, что для любых стратегий $p = (p_1, p_2, \ldots, p_k)$ и $q = (q_1, q_2, \ldots, q_k)$ выполнено

$$\min_j \sum_i m_{ij}p_i \leqslant \max_i \sum_j m_{ij}q_j. \tag{18.3}$$

То, что исходная матрица квадратная, не играло никакой роли в выводе последней формулы, каждая из переменных i и j могла пробегать свое множество значений. Можно даже не нумеровать строки и столбцы исходной матрицы, а дать им какие-то имена. Для матриц с такого рода именованием строк и столбцов чаще используют название *таблица*.

Вернемся теперь от матричных игр к анализу сложности алгоритмов. Пусть имеется конечное множество $\mathscr{A}$ алгоритмов решения некоторой задачи и конечное множество X входов фиксированного размера, и для них составлена числовая таблица, элементы которой проиндексированы элементами множества X (первый индекс) и множества $\mathscr{A}$ (второй индекс). В качестве элемента таблицы с индексами x, A берется величина затрат (например, временных) $C_A(x)$. Если рассмотреть какие-либо распределения вероятностей на X и $\mathscr{A}$, то в силу (18.3) будем иметь

$$\min_{A \in \mathscr{A}} \mathsf{E}_X C_A(x) \leqslant \max_{x \in X} \mathsf{E}_{\mathscr{A}} C_A(x), \tag{18.4}$$

где символами E_X и $\mathsf{E}_{\mathscr{A}}$ обозначены математические ожидания, связанные с заданными (произвольными) распределениями вероятностей на X и $\mathscr{A}$.

Будем рассматривать рандомизированный алгоритм как конечное множество детерминированных («обычных») алгоритмов с некоторым распределением вероятностей на нем, — о возможности такого взгляда на рандомизированные алгоритмы говорилось в конце § 8. Тогда можно определить усредненные затраты этого рандомизированного алгоритма для каждого входа. Фиксируя размер входа, мы можем рассмотреть сложность в худшем случае такого рандомизированного алгоритма $\mathscr{A}$ как максимум усредненных затрат по входам данного размера, эта сложность записана в правой части неравенства (18.4). Левая же часть этого неравенства может быть интерпретирована как наименьшая из сложностей в среднем алгоритмов класса $\mathscr{A}$, для определения этой сложности используется некоторое распределение вероятностей на X.

Мы приходим к принципу, предложенному А. Яо.

Теорема 18.1 (принцип Яо). *Пусть для размера входа зафиксировано некоторое значение n, и пусть все входы размера n образуют конечное множество X, на котором задано некоторое распределение вероятностей. Пусть $\mathscr{A}$ — конечное множество алгоритмов, применимых к элементам множества X, и пусть на $\mathscr{A}$ тоже задано некоторое распределение вероятностей. Множество $\mathscr{A}$ с этим распределением можно рассматривать как единый рандомизированный алгоритм, считая его сложностью $T(n)$ усредненные затраты в худшем случае. Тогда, если найти некоторую нижнюю границу $f(n)$ сложностей в среднем (в соответствии с данным распределением вероятностей на множестве X входов размера n) всех детерминированных алгоритмов класса $\mathscr{A}$, то, независимо от конкретного вида распределений на X и $\mathscr{A}$, будет выполняться неравенство $f(n) \leqslant T(n)$.*

Пример 18.1. Рассмотрим произвольную рандомизированную сортировку массивов фиксированной длины n, которую, по крайней мере теоретически, можно задать как конечное множество детерминированных алгоритмов сортировки массивов длины n с приписанными этим детерминированным алгоритмам вероятностями, в сумме дающими единицу. (Предполагаем, что это вероятностное пространство алгоритмов не зависит от конкретного входа, т. е. конкретного массива длины n.) Тогда в соответствии с принципом Яо сложность этой рандомизированной сортировки не может быть меньше чем $\log_2 n!$, так как при равномерном распределении вероятностей

на Π_n для сложности в среднем детерминированных алгоритмов сортировки мы имеем в силу предложения 17.1 нижнюю границу $\log_2 n!$ при любом n.[1]

Задачи

74. Для обсуждавшихся в задаче 8 алгоритмов сортировки вагонов функция $3n-1$ является нижней границей их сложности, откуда следует, что тот алгоритм, который требуется указать в задаче 8, является оптимальным в рассматриваемом классе.

75. Для сложности класса обсуждавшихся в задаче 17 алгоритмов поиска двери в стене функция n является асимптотической нижней границей, откуда следует, что тот алгоритм, который требуется указать в задаче 17, является оптимальным по порядку сложности в рассматриваемом классе.

76. (Продолжение предыдущей задачи.) Для поиска двери можно указать класс $\mathscr{A}$ алгоритмов, каждый из которых определяется некоторым целым $k \geqslant 2$: путник делает k шагов вправо, и если дверь не найдена, то возвращается назад и делает k шагов влево, и опять возвращается назад, если дверь не найдена; затем делает k^2 шагов вправо, возвращаясь назад в случае неуспеха, затем k^2 шагов влево и т. д. Таким образом, $\mathscr{A} = \{A_2, A_3, \dots\}$, и для сложности по числу шагов имеем $T_{A_k}(n) = 3n$, если $k = n$, и $T_{A_k}(n) > 3n$, если $k \neq n$. Как следствие, в классе $\mathscr{A}$ нет оптимального алгоритма, но каждый алгоритм из этого класса является оптимальным в нем по порядку сложности.

Указание. Пусть m таково, что $k^{m-1} < n \leqslant k^m$. Возможно, что потребуется число шагов, равное $2(k + k^2 + \dots + k^m) + 2(k + k^2 + \dots + k^{m-1}) + n$.

77. Нарисовать дерево быстрой сортировки для случая $n = 3$, считая, что первый элемент всегда выбирается в качестве разбивающего.

78. Дать другое доказательство предложения 14.2. Рассмотреть последовательности из нулей и единиц, соответствующие результатам всех проводимых сравнений в процессе применения сортировки к массивам длины n (каждому массиву сопоставляется своя последовательность).

Указание. Если некоторый алгоритм сортировки требует не более $h(n)$ сравнений, то длина каждой из последовательностей не превосходит $h(n)$, и общее число последовательностей не превосходит $2^{h(n)}$.

[1] Дополнительные примеры применения принципа Яо можно найти в [55, гл. 2].

79. Аналогично предыдущей задаче, дать другое доказательство предложения 14.4.

80. Как уже отмечалось (со ссылкой на приложение D), n является нижней границей сложности как по числу аддитивных операций, так и по числу умножений для класса алгоритмов, вычисляющих значение полинома $a_n x^n + \ldots + a_1 x + a_0$ с помощью операций сложения, вычитания и умножения (при рассмотрении числа n как размера входа $x, a_0, a_1, \ldots, a_n$). Означает ли это, что при вычислении любого фиксированного полинома $p(x)$ степени n при заданном значении переменной x потребуется не менее n аддитивных операций и n умножений? Указать такие нижние границы сложности (по числу аддитивных операций и по числу операций умножения) алгоритмов вычисления полинома $\sum_{k=0}^{n} \binom{n}{k} x^k$, которые растут при $n \to \infty$ существенно медленнее, чем n.

81. Имеется плитка шоколада размера $k \times l$ клеток, которую требуется разломать на отдельные клетки. Каждый разлом применяется к одному куску плитки и производится по прямой линии. Затраты каждого алгоритма решения этой задачи применительно к конкретной плитке измеряются числом потребовавшихся разломов. Указать оптимальный алгоритм решения этой задачи и выписать его сложность. Числа k, l считаются параметрами размера входа алгоритмов.

82. Если условно изобразить элементы массива длины n точками на плоскости, соединяя отрезками те из них, которые непосредственно сравнивались в процессе некоторого алгоритма поиска, — скажем, поиска наименьшего элемента, одновременного поиска наибольшего и наименьшего элементов, поиска k-го по величине элемента (в частности, поиска медианы, т. е. $\left\lceil \frac{n}{2} \right\rceil$-го элемента), — то получающийся граф должен быть связным, иначе мы бы ничего не могли сказать о том, как соотносятся между собой элементы из разных компонент. Показать, что для всех алгоритмов поиска, сопоставляющих указанным (неявным) образом связные графы конкретным массивам, $n - 1$ является нижней границей сложности по числу сравнений (как в худшем случае, так и в среднем); в частности, это верно для алгоритмов поиска медианы.

Указание. Дерево с n вершинами имеет $n - 1$ ребро.

83. Рассмотрим класс алгоритмов поиска k первых (меньших) по величине элементов массива без установления порядка между найденными элементами и класс алгоритмов поиска k-го по величине

элемента. Как обычно, имеется в виду поиск с помощью сравнений. Доказать, что любая нижняя граница сложности по числу сравнений алгоритмов первого класса является нижней границей и для второго класса.

84. Дать доказательство предложения 14.1, построенное по той же схеме, что и доказательство предложения 15.1.

Указание. Рассмотреть тройки множеств (A, B, C), включая в A на каждом этапе алгоритма все те элементы, которые еще не сравнивались, в B — все те, которые участвовали в сравнениях и всегда оказывались меньшими, в C — все те, которые участвовали в сравнениях и в некоторых оказывались бóльшими.

85. Доказать содержащееся в доказательстве предложения 15.1 утверждение, что после первого сравнения постоянно будет выполнено $b \geqslant 1$, $c \geqslant 1$.

86. В классе алгоритмов вычисления a^n с помощью умножений (a фиксировано, $n \in \mathbb{N}$) при рассмотрении n в качестве размера входа существует оптимальный алгоритм. То же самое — при рассмотрении $m = \lambda(n)$ в качестве размера входа.

87. Бинарный алгоритм возведения в степень n не является оптимальным по числу умножений и при использовании $m = \lambda(n)$ в качестве размера входа.

Указание. Рассмотреть алгоритм, который для всех $n \neq 15$ работает как бинарный алгоритм, а при $n = 15$ — несколько быстрее (см. задачу 19), и показать, что такой алгоритм при $m = 4$ имеет сложность меньшую, чем бинарный.

88. (Продолжение задачи 87.) Является ли оптимальным по порядку сложности бинарный алгоритм возведения в степень n при использовании $m = \lambda(n)$ в качестве размера входа?

89. Рассмотренный в задаче 64 алгоритм поиска фальшивой монеты, требующий в худшем случае не более $\lceil \log_3 n \rceil$ взвешиваний, является оптимальным для худшего случая.

90. Нижней границей сложности любого алгоритма деления отрезка на n равных частей с помощью циркуля и линейки (см. задачу 18) при рассмотрении числа n в качестве размера входа является, в частности, $\dfrac{n-1}{2}$.

91. Отсутствие указания основания логарифма в (16.1) является корректным.

92. Функция $\log_2(n+1)$ является нижней границей сложности в среднем алгоритмов поиска места элемента в упорядоченном массиве

длины n с помощью сравнений (считаем, что в диапазоне от 1 до $n+1$ место может быть любым с одинаковой вероятностью $\frac{1}{n+1}$).

Указание. Для доказательства воспользоваться предложением 17.1.

93. (Продолжение предыдущей задачи.) Алгоритм бинарного поиска является оптимальным в среднем по порядку сложности в классе алгоритмов поиска места элемента с помощью сравнений.

94. Алгоритм, описанный в задаче 29, является оптимальным как в худшем случае, так и в среднем.

95. Пусть $\overline{T}_{\mathrm{QS}}(n)$ и $\overline{T}_{\mathrm{opt}}(n)$ — сложности в среднем быстрой сортировки и оптимальной в среднем сортировки. Верно ли, что $\overline{T}_{\mathrm{QS}}(n) \sim \sim \overline{T}_{\mathrm{opt}}(n)$? Если нет, то можно ли подобрать константу c такую, что $\overline{T}_{Q}(n) \sim c\overline{T}_{\mathrm{opt}}(n)$?

96. Утверждение теоремы 17.1 перестает быть верным, если его изменить, удалив условие, что $\tau < \infty$ всюду на рассматриваемом вероятностном пространстве: при выполнении всех остальных условий может оказаться, что $\tau = \infty$ при том, что $\sum\limits_{k=1}^{\infty} h_k$ имеет конечное значение.

Указание. Рассмотреть постоянные величины $\xi_k = \frac{1}{k(k+1)}$, $k = 1, 2, \ldots$, взяв $h_k = \xi_k$ для всех k; при любом $q \geqslant 1$ получается противоречие с измененным утверждением. [1]

97. Функция $\log_2(n+1)$ является нижней границей сложности рандомизированных алгоритмов поиска места элемента в упорядоченном массиве длины n с помощью сравнений. (Предполагается, что каждый из рассматриваемых рандомизированных алгоритмов может быть задан как конечное множество детерминированных алгоритмов поиска места элемента в упорядоченном массиве длины n, с приписанными этим детерминированным алгоритмам вероятностями, в сумме дающими единицу; при этом такое вероятностное пространство алгоритмов не должно зависеть от конкретного входа размера n.)

[1] Эта задача сообщена автору А. В. Бернштейном.

Глава 5

Битовая сложность

§ 19. Битовые операции

Каждый алгоритм строится на основе некоторого фиксированного набора базовых операций, и, согласно определениям из § 1, алгебраическая сложность алгоритма рассматривается при допущении, что затратность операций не зависит от размеров операндов. С позиций компьютерной арифметики это допущение вполне приемлемо, если каждый из операндов умещается в одном слове памяти. Но оно перестает быть приемлемым, если разрешаются операнды любой длины и вычисления проводятся в рамках арифметики многократной точности (арифметики длинных чисел). Здесь анализ сложности должен исходить из других принципов.

Приемлемой является, например, следующая модель как система допущений, принимаемых нами при анализе алгоритмов. Число представляется набором бит, являющихся содержимым его двоичных разрядов (знак числа — тоже бит, например, обычно знаки + и − изображаются соответственно нулем и единицей), и все арифметические операции сводятся, в конечном счете, к битовым операциям. В качестве битовых операций могут выступать булевы операции $\vee$, $\wedge$, $\neg$ (другая нотация: or, and, not) при интерпретации 1 = «истина», 0 = «ложь» встречающихся бит; эти операции считаются равнозатратными. Все биты, участвующие в записи числа, считаются равнодоступными.

Можно смотреть на все это так, что ячейки памяти машины содержат по одному биту каждая, а в остальном действует принцип PAM — ячеек бесконечно много и они в любой момент равнодоступны. Это — *битовая модель вычислений* (соответствующее сокращение БМ может расшифровываться как «битовая модель» и как «битовая машина»).

Для анализа сложности с использованием БМ нет необходимости переписывать алгоритмы, детализируя их до уровня битовых опера-

ций. Вместо этого можно рассматривать привлекаемые алгоритмом базовые арифметические операции как основанные на битовых вычислениях процедуры, полагая временные и пространственные затраты алгоритма равными числу затрачиваемых битовых операций и, соответственно, требующихся дополнительных бит памяти. В качестве размера входа часто рассматривается либо суммарная битовая длина всего входа (всех входных данных), либо максимум этих длин. В некоторых случаях вместо битовых длин целых чисел берутся сами эти числа. Возможно и использование нескольких параметров размера входа.

Определение 19.1. Пусть выбран какой-то размер входа. Рассмотрение каждой из базовых операций алгоритма A как процедуры, основанной на битовых вычислениях, и измерение затрат на выполнение A для каждого конкретного входа в битовых операциях или, соответственно, в хранимых дополнительных битах, приводит к *битовой сложности* (временной или пространственной) алгоритма A.

Для нахождения битовой сложности какого-либо алгоритма, построенного на основных арифметических операциях над целыми числами, полезно знать битовые сложности самих основных арифметических операций. Мы исследуем школьные алгоритмы сложения, вычитания, умножения и деления «столбиком» неотрицательных целых чисел в двоичной системе.

В процессе сложения «столбиком» мы шаг за шагом вычисляем двоичные цифры суммы, продвигаясь от младших разрядов к старшим. При этом в старшие разряды каждый раз переносится 0 или 1. Таким образом, на каждом шаге мы работаем с тремя битами (на начальном шаге, когда рассматриваются самые младшие разряды, бит переноса полагаем равным нулю). Для сложения многозначных чисел нам нужны две битовые процедуры трех аргументов (два аргумента — это цифры одноименных разрядов двух данных чисел, третий — это бит переноса из предшествующих разрядов), одна из процедур дает цифру соответствующего разряда суммы, вторая — новый бит переноса в старшие разряды. Мы не станем рассматривать этот вопрос более подробно, потому что и без деталей ясно, что затраты по построению каждого разряда суммы ограничены сверху некоторой константой.

Алгоритм сложения «столбиком» будем обозначать буквами Add, от английского слова addition — сложение. Обозначим через a и b операнды алгоритма ($a, b \in \mathbb{N}^+$), через m_1, m_2 — их битовые длины: $m_1 = \lambda(a)$, $m_2 = \lambda(b)$.

Лемма 19.1. *Количество $C^T_{\mathrm{Add}}(a,b)$ битовых операций, затрачиваемых при сложении a и b «столбиком», удовлетворяет неравенствам*

$$\min\{m_1,m_2\} \leqslant C^T_{\mathrm{Add}}(a,b) \leqslant c(\max\{m_1,m_2\}+1), \tag{19.1}$$

где c — некоторая положительная константа.

Доказательство. Принимая во внимание замечание относительно ограниченности затрат на построение каждого разряда суммы, а также то, что битовая длина суммы $a+b$ не может превышать $\max\{m_1,m_2\}+1$, получаем оценку $C^T_{\mathrm{Add}}(a,b) \leqslant c(\max\{m_1,m_2\}+1)$. Пусть $m' = \min\{m_1,m_2\}$. Очевидно, что m' младших цифр суммы $a+b$ получаются выполнением битовых операций над соответствующими цифрами обоих слагаемых и битами переноса (в то время как часть цифр старших разрядов большего из слагаемых может быть просто перенесена в старшие разряды суммы). Это дает нам неравенство $\min\{m_1,m_2\} \leqslant C^T_{\mathrm{Add}}(a,b)$. □

В дальнейшем будем полагать

$$m = \max\{m_1,m_2\}. \tag{19.2}$$

Предложение 19.1. *Пусть $T^*_{\mathrm{Add}}(m)$ — сложность по числу битовых операций алгоритма сложения «столбиком» при использовании m в качестве размера входа. Тогда $T^*_{\mathrm{Add}}(m) = \Theta(m)$.*

Доказательство. При фиксированном m мы можем выбрать a и b так, что $m_1 = m_2 = m$, тогда $\min\{m_1,m_2\} = \max\{m_1,m_2\} = m$; далее применяем лемму 19.1. □

Алгоритм сложения «столбиком» позволяет получать сумму a и b на месте максимального из этих двух чисел, в этом случае для битовой пространственной сложности имеем $S^*_{\mathrm{Add}}(m) = O(1)$. Если для суммы чисел используется дополнительное место (чтобы не изменять данные слагаемые), то, соответственно, $S^*_{\mathrm{Add}}(m) = m + O(1)$.

Утверждения леммы 19.1 и предложения 19.1 верны, как легко убедиться, и для операции вычитания (при вычитании «столбиком» в старших разрядах занимается 0 или 1; битовая длина разности не превосходит m).

Формула $T^*_{\mathrm{Add}}(m) = \Theta(m)$ говорит о том, что алгоритм сложения «столбиком» при рассмотрении m в качестве размера входа является оптимальным по порядку битовой сложности: мы не можем игнорировать содержимое разрядов, и поэтому m является нижней границей битовой сложности алгоритмов сложения. Для алгоритмов умножения и деления «столбиком» в дальнейшем будет показано, что их

сложность имеет порядок m^2, и в этой ситуации уже ниоткуда не следует, что не может существовать алгоритмов с лучшей асимптотикой сложности, и такие алгоритмы действительно существуют (мы об этом еще будем говорить). Поэтому умножение и деление «столбиком» мы будем называть, как это делается в некоторых книгах, *наивным* умножением и делением соответственно, используя обозначения NM и ND (от английских слов naive multiplication/division — наивное умножение/деление). Сложение «столбиком» будем просто называть сложением.

Пример 19.1. Допустим, что для умножения двух целых положительных чисел a и b мы используем сложения:

$$a+a, \quad (a+a)+a, \quad \ldots, \quad (\underbrace{a+\ldots+a}_{b-1})+a. \tag{19.3}$$

Исследуем битовую сложность такого «сверхнаивного» умножения, считая m размером входа. В силу леммы 19.1 при $m_1 = m_2 = m$ на каждое сложение уйдет не менее m битовых операций. Учитывая, что $b > 2^{m-1}$, получаем для исследуемой сложности оценку $\Omega(2^m m)$. С другой стороны, $ab < 2^{2m}$, поэтому результат каждого шага в (19.3) имеет не превосходящую $2m$ битовую длину. Это означает, что число битовых операций на каждом из шагов не превосходит $c \cdot 2m$, где c — константа. Отсюда получаем оценку $O(2^m m)$, и в итоге — оценку $\Theta(2^m m)$.

Наряду с размером входа (19.2) часто рассматривают два параметра m_1, m_2 размера входа.

Предложение 19.2. *Если рассмотреть два параметра $m_1 = \lambda(a)$, $m_2 = \lambda(b)$ размера входа, то битовая сложность $T^{**}_{\mathrm{Add}}(m_1, m_2)$ алгоритма сложения будет допускать оценку $\Theta(\max\{m_1, m_2\})$.*

Доказательство. Пусть, например, $\max\{m_1, m_2\} = m_1$. Тогда оценка $T^{**}_{\mathrm{Add}}(m_1, m_2) = \Omega(\max\{m_1, m_2\})$ подтверждается наличием входа $a = 2^{m_1} - 1$, $b = 2^{m_2} - 1$. Оценка же $T^{**}_{\mathrm{Add}}(m_1, m_2) = O(\max\{m_1, m_2\})$ следует из леммы 19.1. □

Битовая сложность рассматривается не только в связи с арифметическими задачами. Например, булевы матрицы смежности, которые служат одним из стандартных средств представления графов без кратных ребер, являются битовыми матрицами, и целому ряду алгоритмов решения задач на графах естественным образом сопоставляется битовая сложность. Об этом будет идти разговор в § 23. Но прежде,

в § 20 и 21, мы займемся битовой сложностью наивного умножения и деления.

Еще одно замечание в заключение этого параграфа. Битовый анализ может учесть мельчайшие затраты, связанные с выполнением алгоритма. Вопрос в том, нужен ли настолько детализированный подход, коль скоро компьютеры выполняют операции над основными структурами данных на уровне слов, длина же слова — это, как правило, 64 бита. В связи с этим иногда вместо битовой рассматривается так называемая *словесная* сложность [1]. Но принципиальной разницы между битовой и словесной сложностью нет. Можно сказать так, что битовая сложность предполагает использование системы счисления с основанием 2, а словесная — с основанием 64. Анализ словесной сложности не является, в сравнении с анализом битовой сложности, ни более легким, ни более информативным.

§ 20. Наивная арифметика: умножение

Предложение 20.1. *Пусть $T^{*}_{\mathrm{NM}}(m)$ — временная битовая сложность наивного умножения при использовании m в качестве размера входа, а $T^{**}_{\mathrm{NM}}(m_1, m_2)$ — временная битовая сложность этого же алгоритма при использовании m_1, m_2 в качестве двух параметров размера входа. Тогда*

$$T^{*}_{\mathrm{NM}}(m) = \Theta(m^2), \quad T^{**}_{\mathrm{NM}}(m_1, m_2) = \Theta(m_1 m_2). \tag{20.1}$$

Доказательство. Затраты наивного умножения a на b связаны с суммированием m_2 чисел $n_1, n_2, \ldots, n_{m_2}$, где каждое n_i равно 0 или $a \cdot 2^{i-1}$ в зависимости от соответствующей цифры в двоичной записи b. Несложной индукцией доказывается, что для $s_i = n_1 + n_2 + \ldots + n_i$, $i = 1, 2, \ldots, m_2$, выполнено $\lambda(s_i) \leqslant m_1 + i$. Если $n_{i+1} \neq 0$, то последние i цифр числа n_{i+1} суть нули, и при вычислении $s_{i+1} = s_i + n_{i+1}$ мы не притрагиваемся к этим цифрам, а преобразуем s_i в s_{i+1} сложением двух чисел, первое из которых образовано всеми разрядами числа s_i, кроме i последних (согласно сказанному, битовая длина этого числа не превосходит m_1), второе же слагаемое есть a, и его битовая длина равна m_1. Число битовых операций, требующихся для преобразования s_i в s_{i+1}, не превосходит $c(m_1 + 1)$ для некоторой положительной константы c. В том случае, когда все цифры числа b равны 1, это число при любом i не меньше, чем m_1, по лемме 19.1.

[1] См., например, книгу [50]. В литературе на английском языке, в частности в книге [50], используется термин «word complexity».

Поэтому битовые затраты, связанные с наивным умножением a на b, не превосходят $c(m_1+1)m_2$, а при $b=2^{m_2}-1$ (двоичная запись этого b состоит только из единиц) затраты не меньше, чем m_1m_2. Это дает оценку $T^{**}_{\mathrm{NM}}(m_1,m_2)=\Theta(m_1m_2)$.

Рассмотрим теперь m как размер входа. Так как $m_1\leqslant m$ и $m_2\leqslant m$, то затраты не превосходят $c(m+1)m$, это дает оценку $T^{*}_{\mathrm{NM}}(m)=O(m^2)$. С другой стороны, если $m_1=m_2=m$ и $a=b=2^m-1$, то затраты не могут быть меньше, чем m^2, и, следовательно, $T^{*}_{\mathrm{NM}}(m)=\Omega(m^2)$. Таким образом, $T^{*}_{\mathrm{NM}}(m)=\Theta(m^2)$. □

Для пространственной битовой сложности наивного умножения мы имеем, очевидно, оценки

$$S^{*}_{\mathrm{NM}}(m)=2m+O(1),\quad S^{**}_{\mathrm{NM}}(m_1,m_2)=m_1+m_2+O(1).$$

Обсудим переход от параметров m_1,m_2 размера входа к параметрам a,b (можно считать, что и к параметрам $\log a,\log b$ — между a,b и $\log a,\log b$ в этом смысле нет принципиальной разницы, так как одни параметры однозначно определяют другие; в то же время мы не можем восстановить a,b по $\lceil\log_2(a+1)\rceil$, $\lceil\log_2(b+1)\rceil$).

Для перехода в верхних оценках сложности от $\lambda(k)$ к $\log_2 k$, $k\in\mathbb{N}^*$, часто оказывается удобным простое неравенство $\lceil\log_2(k+1)\rceil\leqslant 2\log_2 k$, справедливое для всех $k\geqslant 2$:

$$\lceil\log_2(k+1)\rceil=\lfloor\log_2 k\rfloor+1\leqslant\log_2 k+1\leqslant 2\log_2 k.$$

Поэтому, например,

$$m_1\leqslant 2\log_2 a,\quad m_2\leqslant 2\log_2 b$$

и

$$m_1m_2\leqslant 4\log_2 a\log_2 b \tag{20.2}$$

для всех $a,b\geqslant 2$. Имеем для достаточно больших m_1,m_2 и некоторой положительной константы c

$$C^{T}_{\mathrm{NM}}(a,b)\leqslant T^{**}_{\mathrm{NM}}(m_1,m_2)\leqslant cm_1m_2\leqslant 4c\log_2 a\log_2 b.$$

Мы можем написать $C^{T}_{\mathrm{NM}}(a,b)=O(\log a\log b)$, считая, что выражение под знаком O рассматривается как функция двух переменных a,b, при этом $a,b\to\infty$; считаем также, что логарифмы имеют общее основание (подобные предположения будут подразумеваться и в дальнейшем). Остается заметить, что битовая временная сложность при использовании самих a,b в качестве параметров размера входа совпадает с битовыми временными затратами. Это позволяет получить из оценки $T^{**}_{\mathrm{NM}}(m_1,m_2)=O(m_1m_2)$ оценку $T_{\mathrm{NM}}(a,b)=O(\log a\log b)$ для

временной битовой сложности при использовании двух параметров размера входа a, b. Итак:

При использовании самих положительных целых a, b в качестве параметров размера входа сложность наивного умножения a на b по числу битовых операций допускает оценку $O(\log a \log b)$.

Выше наивное умножение (умножение «столбиком») понималось так, что если какая-то цифра числа b равна нулю, то, несмотря на это, построение соответствующих n_i и s_i требует не менее m_1 битовых операций. При таком взгляде сложность будет величиной $\Theta(\log a \log b)$. Но можно считать, что в рассматриваемом случае битовые затраты на получение s_i не зависят от a и ограничены константой. Тогда оценка $\Omega(\log a \log b)$ места не имеет: если $a = 2^{k_1}$, $b = 2^{k_2}$, где k_1, k_2 — положительные целые, то битовые затраты будут ограничены линейной функцией от k_1, k_2.

Пример 20.1. Покажем, что битовая временная сложность алгоритма вычисления $n!$ с помощью пошаговых наивных умножений

$$2 \cdot 3, \quad (2 \cdot 3) \cdot 4, \quad \ldots, \quad (2 \cdot 3 \ldots (n-1)) \cdot n$$

при использовании n в качестве размера входа допускает верхнюю оценку $O((n \log n)^2)$. В силу предложения 20.1 и неравенства (20.2) рассматриваемая сложность не превосходит $cf(n)$, где c — некоторая положительная константа, а значение $f(n)$ равно

$$\log_2 2 \log_2 3 + \log_2(2 \cdot 3) \log_2 4 + \ldots + \log_2(2 \cdot 3 \ldots (n-1)) \log_2 n.$$

Имеем

$$f(n) \leqslant \log_2(2 \cdot 3 \ldots (n-1))(\log_2 2 + \log_2 3 + \ldots + \log_2 n) \leqslant \log_2^2(n!).$$

Одним из следствий формулы Стирлинга является оценка $\log_2(n!) = O(n \log n)$, откуда $\log_2^2(n!) = O((n \log n)^2)$, и $f(n) = O((n \log n)^2)$.

Пример 20.2. Мы упоминали о том, что для алгоритма, входом которого является несколько целых чисел, можно в некоторых случаях определять размер входа как суммарную битовую длину всех этих чисел. Предположим, что имеется несколько целых чисел $a_1, a_2, \ldots, a_n$, каждое из которых $\geqslant 2$, и с помощью пошаговых наивных умножений

$$a_1 a_2, \quad (a_1 a_2) a_3, \quad \ldots, \quad (a_1 a_2 \ldots a_{n-1}) a_n \tag{20.3}$$

вычисляется произведение $A = a_1 a_2 \ldots a_n$. Пусть суммарная битовая длина чисел равна M, и число M рассматривается как размер входа алгоритма (20.3). Покажем, что тогда битовая сложность этого

алгоритма допускает верхнюю оценку $O(M^2)$, — примечательно, что число n, т. е. общее количество сомножителей, в этой оценке не появляется.

В силу предложения 20.1 и неравенства (20.2) битовые затраты на вычисление $a_1, a_2, \ldots a_n$ не превосходит $cF(a_1, a_2, \ldots, a_n)$, где c — некоторая положительная константа, а значение $F(a_1, a_2, \ldots, a_n)$ равно

$$\log_2 a_1 \log_2 a_2 + \log_2(a_1 a_2) \log_2 a_3 + \ldots + \log_2(a_1 a_2 \ldots a_{n-1}) \log_2 a_n$$

(использовано предположение, что $a_1, a_2, \ldots, a_n \geqslant 2$). Мы легко получаем

$$F(a_1, a_2, \ldots, a_n) \leqslant \log_2(a_1 \ldots a_{n-1})(\log_2 a_2 + \ldots + \log_2 a_n)$$

и

$$F(a_1, a_2, \ldots, a_n) \leqslant (\log_2 a_1 + \log_2 a_2 + \ldots + \log_2 a_n)^2.$$

Последнее неравенство позволяет перейти к рассмотрению M в качестве размера входа, так как, очевидно,

$$F(a_1, a_2, \ldots, a_n) \leqslant \big(\lceil \log_2(a_1+1) \rceil + \lceil \log_2(a_2+1) \rceil + \ldots + \lceil \log_2(a_n+1) \rceil\big)^2 = M^2,$$

что дает нам требуемое.

Доказанное утверждение справедливо и в том случае, когда среди $a_1, a_2, \ldots, a_n$ имеются единицы, если считать, что умножение на 1 требует одной битовой операции. Если среди $a_1, a_2, \ldots, a_n$ имеются k единиц, то из $M \geqslant k$ следует $(M-k)^2 + k \leqslant M^2$.

§ 21. Наивная арифметика: деление с остатком

Обратимся к наивному делению с остатком. Здесь мы считаем, что $m = m_1 \geqslant m_2$.

Предложение 21.1. *Пусть $T^*_{\mathrm{ND}}(m)$ — сложность по числу битовых операций наивного деления при использовании m в качестве размера входа. Пусть $T^{**}_{\mathrm{ND}}(m_1, m_2)$ — сложность по числу битовых операций этого же алгоритма при использовании m_1, m_2 в качестве двух параметров размера входа. Тогда*

$$T^*_{\mathrm{ND}}(m) = \Theta(m^2), \quad T^{**}_{\mathrm{ND}}(m_1, m_2) = \Theta((m_1 - m_2 + 1)m_2). \tag{21.1}$$

Доказательство. Наивное деление a на b сводится к ряду вычитаний числа b, и в каждом вычитании битовая длина уменьшаемого либо равна, либо на единицу превосходит m_2. Битовые затраты на каждое такое вычитание не могут превзойти, как мы знаем, $c(m_2 + 1)$. Количество всех таких вычитаний не превосходит $m_1 - m_2 + 1$, откуда $T^{**}_{\mathrm{ND}}(m_1, m_2) = O((m_1 - m_2 + 1)(m_2 + 1))$ и, после

упрощения, $T^{**}_{\mathrm{ND}}(m_1, m_2) = O((m_1 - m_2 + 1)m_2)$ (см. задачу 101). С другой стороны, если, например, $a = 2^{m_1} - 1$, $b = 2^{m_2 - 1}$, то число вычитаний равно $m_1 - m_2 + 1$, и, как следствие, битовые затраты на наивное деление будут не меньше, чем $(m_1 - m_2 + 1)m_2$. Поэтому $T^{**}_{\mathrm{ND}}(m_1, m_2) = \Theta((m_1 - m_2 + 1)m_2)$.

Оценка $T^{*}_{\mathrm{ND}}(m) = O(m^2)$ следует теперь из неравенства $m^2 \geqslant m_1 m_2 \geqslant (m_1 - m_2 + 1)m_2$. Если взять $a = 2^m - 1$, $b = 2^{\lfloor m/2 \rfloor}$, то битовые затраты на наивное деление будут не меньше чем $\left(m - \left\lfloor \frac{m}{2} \right\rfloor + 1\right)\left(\left\lfloor \frac{m}{2} \right\rfloor + 1\right)$, и, как следствие, $T^{*}_{\mathrm{ND}}(m) = \Omega(m^2)$. Получаем $T^{*}_{\mathrm{ND}}(m) = \Theta(m^2)$. □

Мы не пишем $T^{**}_{\mathrm{ND}}(m_1, m_2) = \Theta((m_1 - m_2)m_2)$ из-за возможности равенства $m_1 = m_2$ (из которого не следует, что $a = b$). Эта возможность указывает и на то, что неверно соотношение $T^{**}_{\mathrm{ND}}(m_1, m_2) = \Theta(m_1 m_2)$.

Предложение 21.2. *При использовании самих положительных целых a, b, $a \geqslant b > 1$, в качестве параметров размера входа, сложность наивного деления a на b допускает оценку $O(\log a \log b)$. При использовании a в качестве размера входа имеем оценку $O(\log^2 a)$.*

Доказательство. Как уже говорилось, число битовых операций, требуемых наивным умножением, не превосходит произведения

$$c\big((\lfloor \log_2 a \rfloor + 1) - (\lfloor \log_2 b \rfloor + 1) + 1\big)(\lfloor \log_2 b \rfloor + 1)$$

(c — положительная константа), которое не превосходит в свою очередь

$$c(\log_2 a - \log_2 b + 2)(\log_2 b + 1).$$

Каждое слагаемое, возникающее в результате раскрытия скобок в последнем выражении, есть $O(\log a \log b)$ при $a, b \to \infty$, $a \geqslant b$. Отсюда следует первая часть утверждения. Вторая часть следует из того, что $\log_2 a \log_2 b \leqslant \log_2^2 a$ при $a \geqslant b$. □

Пример 21.1. Пусть n и k — положительные целые числа, $k \geqslant 2$, заданные в двоичной системе счисления. Покажем, что при избрании n в качестве размера входа n, k (равно как и при избрании n, k в качестве двух параметров размера этого входа) битовая сложность обычного алгоритма построения k-ичной записи числа n с помощью наивных делений с остатком допускает оценку $O(\log^2 n)$.

В самом деле, при построении k-ичной записи n шаг за шагом рассматриваются числа $q_0, q_1, \ldots, q_t$, $t = \lfloor \log_k n \rfloor + 1$, где $q_0 = n$, $q_i = \left\lfloor \frac{q_{i-1}}{k} \right\rfloor$, $i = 1, 2, \ldots, t$. Очевидно, что $q_i \leqslant n$, в силу чего общие затраты всех шагов для данных n и k допускают оценку $O(\log n \log k \log_k n)$. Очевидно, $\log k \log_k n = \log n$ (например, $\log_2 k \log_k n = \log_2 n$), поэтому оценка для

затрат переписывается в виде $O(\log^2 n)$; при указанных выше размерах входа эта оценка является и оценкой сложности.

Вновь вернемся к обозначению m для битовой длины максимального из двух данных целых положительных чисел, и пусть m рассматривается как размер входа a, b алгоритмов умножения и деления с остатком. Для сложностей наивного умножения и деления мы получили оценки $\Theta(m^2)$, следствием которых являются нижние оценки $\Omega(m^2)$. Вместе с этим, для умножения и деления известны алгоритмы, сложности которых при $m \to \infty$ растут существенно медленнее, чем m^2: первый алгоритм такого рода для умножения был предложен в 1962 г. А. А. Карацубой (верхняя оценка $O(m^{\log_2 3})$), наилучшую из известных к настоящему времени верхнюю асимптотическую оценку битовой сложности $O(m \log m \log\log m)$ имеет алгоритм А. Шенхаге и Ф. Штрассена. Существуют алгоритмы деления, сложность которых допускает такие же оценки (см. задачу 152 к главе 7). Алгоритм Карацубы будет рассмотрен нами в § 27, алгоритм Шенхаге—Штрассена в этом курсе подробно не рассматривается; несколько слов о нем будет сказано в конце § 27.

Пример 21.2. Исследуем битовую сложность алгоритма Евклида. В качестве размера входа можно рассмотреть большее число a_0, но ни в коем случае не a_1: если фиксировано a_1, то, неограниченно увеличивая a_0, мы будем неограниченно увеличивать битовые затраты, связанные с первым делением с остатком (a_0 на a_1). Поэтому при выборе a_1 в качестве размера входа битовая сложность этого алгоритма тождественно равна бесконечности — здесь битовая сложность существенно отличается от алгебраической (т. е. в данном случае от сложности по числу делений). Можно считать a_0 размером входа, а можно рассмотреть два параметра a_0, a_1 размера входа. Если m_0, m_1 — битовые длины данных чисел a_0, a_1, то в качестве размера входа можно рассмотреть $m = \max\{m_0, m_1\} = m_0$ или же два параметра входа m_0, m_1. В настоящем примере мы будем использовать m.

Прежде всего покажем, что для алгоритма Евклида

$$a_{i-1} = q_i a_i + a_{i+1}, \quad i = 1, 2, \ldots, n, \quad a_{n+1} = 0,$$

выполняется

$$a_0 \geqslant q_1 q_2 \ldots q_n. \tag{21.2}$$

В самом деле,

$$a_0 > q_1 a_1, \quad a_1 > q_2 a_2, \quad \ldots, \quad a_{n-2} > q_{n-1} a_{n-1}, \quad a_{n-1} = q_n a_n,$$

и получаем $a_0 \geqslant q_1 q_2 \ldots q_n a_n$, откуда следует (21.2).

Для построения a_{i+1} делением a_{i-1} на a_i с остатком требуется не более $c(\lfloor\log_2 q_i\rfloor+1)(\lfloor\log_2 a_i\rfloor+2)$ битовых операций, где c — положительная константа. Это дает общую оценку сверху

$$c\sum_{i=1}^{n}(\lfloor\log_2 q_i\rfloor+1)(\lfloor\log_2 a_i\rfloor+2),$$

при этом возможно, что некоторые из q_i равны единице. Написанная сумма не превосходит

$$c(\lfloor\log_2 a_1\rfloor+2)\sum_{i=1}^{n}(\lfloor\log_2 q_i\rfloor+1).$$

При $a_1>1$ выполнено $\lfloor\log_2 a_1\rfloor+2\leqslant 3\log_2 a_1$ и, как следствие,

$$c(\lfloor\log_2 a_1\rfloor+1)\sum_{i=1}^{n}(\lfloor\log_2 q_i\rfloor+2)\leqslant 3c(\log_2 a_1)\left(n+\sum_{i=1}^{n}\log_2 q_i\right)=$$
$$=3c(\log_2 a_1)(n+\log_2(q_1q_2\ldots q_n))\leqslant 3c(\log_2 a_1)(n+\log_2 a_0).$$

Как мы знаем, $n\leqslant 2\log_2 a_1+1$; при $a_1>1$, очевидно, можем написать $n\leqslant 3\log_2 a_1$. Это дает нам оценку сверху

$$3c(\log_2 a_1)(3\log_2 a_1+\log_2 a_0)$$

для числа битовых операций при $a_1>1$. Учитывая, что $a_0\geqslant a_1$, мы получаем отсюда следующее.

Количество битовых операций, затрачиваемых при применении алгоритма Евклида к a_0, a_1, допускает оценку

$$O(\log a_0\log a_1) \tag{21.3}$$

и оценки $O(\log^2 a_0)$, $O(m^2)$, $m=\lambda(a_0)$.

Приведенные оценки получены в предположении, что для построения остатков используется наивное деление, имеющее квадратичную сложность. Может ли быть так, что привлечение имеющего меньшую битовую сложность алгоритма деления с остатком приведет к существенно меньшим битовым затратам алгоритма Евклида? Ответ отрицательный: нетрудно, например, показать, что если a_0 берется в качестве размера входа, то для битовой сложности алгоритма Евклида выполняется оценка $\Omega(\log^2 a_0)$.

В самом деле, в примере 10.1 было показано, что для каждого a_0 можно подобрать меньшее его a_1 такое, что для числа делений с остатком выполняется неравенство (10.10). Если обозначить это число делений через n, то будем иметь $n=\Omega(\log a_0)$, при этом

$$\lambda(a_0),\lambda(a_1),\ldots,\lambda(a_n)$$

является невозрастающей конечной последовательностью натуральных чисел, и в силу предложения 9.1 никакое значение не встречается в ней более двух раз. Так как деление с остатком a_{i-1} на a_i требует не менее $\lambda(a_i)$ битовых операций, то общее число битовых операций не может быть меньше, чем $1+1+2+2+\ldots+k+k=k(k+1)$ при $k=\left\lfloor\frac{n+1}{2}\right\rfloor$ и $n=\Omega(\log a_0)$. Следовательно, общее число битовых операций есть $\Omega(\log^2 a_0)$.

Вместе с ранее установленной оценкой $O(\log^2 a_0)$ это дает оценку $\Theta(\log^2 a_0)$. Теорема 4.2 из § 4 приводит нас к оценке $\Theta(m^2)$ для битовой сложности алгоритма Евклида при избрании битовой длины m числа a_0 в качестве размера входа. Итак:

Если алгоритм Евклида основывается на некотором алгоритме деления с остатком, битовые затраты которого применительно к делимому v и делителю w допускают верхнюю оценку $O(\log v \log w)$, то при рассмотрении большего входного числа a_0 в качестве размера входа битовая сложность алгоритма Евклида допускает оценку $\Theta(\log^2 a_0)$. При рассмотрении $m=\lambda(a_0)$ в качестве размера входа имеем оценку $\Theta(m^2)$.

Оказывается, что полученные оценки сохраняют силу и при рассмотрении расширенного алгоритма Евклида (пример 9.1) — обстоятельство, имеющее большое значение для модулярной арифметики (§ 22).

Предложение 21.3. *Пусть расширенный алгоритм Евклида основывается на некоторых алгоритмах деления с остатком и умножения, битовые затраты каждого из которых применительно к целым v и w допускают верхнюю оценку $O(\log v \log w)$. Тогда, при рассмотрении большего входного числа a_0 в качестве размера входа, битовая сложность расширенного алгоритма Евклида допускает оценку $\Theta(\log^2 a_0)$, а при рассмотрении битовой длины m большего числа a_0 в качестве размера входа — оценку $\Theta(m^2)$.*

Доказательство. Исходя из вычислительных формул (9.8) и принимая во внимание (9.12), (9.13), мы заключаем, что битовые затраты, дополнительные к затратам собственно алгоритма Евклида («нерасширенного») на вычисление s_n, t_n, допускают оценку $O(\log a_0 \log a_1)$ — это обосновывается так же, как оценка (21.3). Поэтому сложность добавочной части расширенного алгоритма Евклида допускает оценку $O(\log^2 a_0)$ и, в силу теоремы 4.2, оценку $O(m^2)$. Остается заметить, что, например, $\Theta(m^2)+O(m^2)=\Theta(m^2)$. □

§ 22. Модулярная арифметика

Многие арифметические алгоритмы основываются на вычислениях по некоторому целому модулю k, или, как говорят, на *модулярных вычислениях*; соответственно, говорят о *модулярной арифметике*. Далее в этом параграфе мы считаем k фиксированным целым положительным числом.

Целые a и b называют *сравнимыми по модулю* k и пишут

$$a \equiv b \pmod{k}, \tag{22.1}$$

если $k \mid (a-b)$, т. е. если k является делителем разности $a-b$. Соотношение вида (22.1) называется *сравнением*. Изложение основ теории сравнений можно найти в любом учебнике алгебры[1]. Мы приведем основные элементарные факты этой теории, не останавливаясь на их доказательстве:

(i) *бинарное отношение сравнимости по модулю k является отношением эквивалентности на множестве целых чисел $\mathbb{Z}$*;

(ii) *если*

$$a_1 \equiv b_1 \pmod{k} \quad \textit{и} \quad a_2 \equiv b_2 \pmod{k},$$

то

$$a_1 \pm a_2 \equiv b_1 \pm b_2 \pmod{k} \quad \textit{и} \quad a_1 a_2 \equiv b_1 b_2 \pmod{k};$$

(iii) *каждое целое число a сравнимо по модулю k в точности с одним целым числом из множества* $\{0, 1, \ldots, k-1\}$.

В силу (i) множество $\mathbb{Z}$ разбивается на классы эквивалентности по модулю k (кратко: классы по модулю k), и, согласно (ii), эти классы образуют кольцо, если считать, что сумма двух классов — это класс, содержащий сумму каких-либо чисел, взятых по одному из складываемых классов, а произведение — это, соответственно, класс, содержащий произведение каких-либо чисел, взятых по одному из перемножаемых классов. Нулем этого кольца является класс, содержащий число 0. Для введенного кольца используется обозначение $\mathbb{Z}_k$, его называют кольцом *вычетов по модулю* k (вычет — это в данном случае другое название класса эквивалентности по модулю k).

Любое множество $\{a_0, a_1, \ldots, a_{k-1}\}$ чисел, взятых по одному из каждого класса по модулю k, называется *полной системой представителей* по модулю k.

Множество

$$\mathbb{I}_k = \{0, 1, \ldots, k-1\},$$

[1] См., например, [14, § 42, 43] и [22, гл. 4, § 4, п. 2].

указанное в (iii), называется *канонической* (или *начальной*) полной системой представителей по модулю k. Иногда используется и *симметричная* полная система представителей:

$$\mathbb{S}_k = \left\{ \left\lceil \frac{-k+1}{2} \right\rceil, \ldots, -1, 0, 1, \ldots, \left\lfloor \frac{k}{2} \right\rfloor \right\},$$

которая является симметричной (при условии игнорирования знаков) только при нечетном k: например, $\mathbb{S}_3 = \{-1, 0, 1\}$ и $\mathbb{S}_4 = \{-1, 0, 1, 2\}$.

Пусть для изображения элементов кольца $\mathbb{Z}_k$ используется система $\mathbb{I}_k$. Операциям сложения и умножения в $\mathbb{Z}_k$ соответствуют обычные сложение и умножение в $\mathbb{Z}$ с последующей заменой полученного результата остатком от его деления на k. Если некоторому элементу кольца $\mathbb{Z}_k$ соответствует число a системы $\mathbb{I}_k$, то противоположному (обратному по сложению) элементу будет соответствовать число $k - a$, если $a \neq 0$, и число 0, если $a = 0$.

Исследуем битовую сложность операций в кольце $\mathbb{Z}_k$. Будем считать размером входа битовую длину m числа k и ограничимся верхними асимптотическими оценками. Для операций сложения и нахождения противоположной величины имеем, очевидно, оценку $O(m)$. Использование операции наивного умножения целых чисел приводит к оценке $O(m^2)$ для сложности умножения в $\mathbb{Z}_k$. Эта оценка, разумеется, сохраняется и при использовании более быстрого умножения и деления в $\mathbb{Z}$.

Как известно, в случае простого k кольцо $\mathbb{Z}_k$ является полем. При составном k это кольцо полем не является и, более того, содержит делители нуля: если $k = pq$, $p > 1, q > 1$, то произведение элементов $\mathbb{Z}_k$, изображаемых числами $p, q \in \mathbb{I}_k$, равно нулю. Допустим, что k является простым и примем для этого простого числа обозначение p, по-прежнему считая его битовую длину равной m. Нахождение обратного к элементу кольца $\mathbb{Z}_p$, представленному целым ненулевым числом $a \in \mathbb{I}_p$, может быть выполнено с помощью расширенного алгоритма Евклида. Так как p и a, очевидно, взаимно просты, то с помощью расширенного алгоритма Евклида можно найти s и t такие, что $sp + ta = 1$. Получаем $ta \equiv 1 \pmod{p}$, т. е. обратным к классу, содержащему a, является класс, содержащий t. В силу (9.13) имеем $|t| < p$; если $t < 0$, то заменяем t на $t + p$ (напомним, что мы используем элементы системы $\mathbb{I}_p$ для изображения элементов поля $\mathbb{Z}_p$).

Пример 22.1. Для $p = 13, a = 5$ получаем с помощью расширенного алгоритма Евклида $2 \cdot 13 + (-5) \cdot 5 = 1$, и отсюда $8 = -5 + 13$ является обратным для 5 в $\mathbb{Z}_{13}$ (проверка подтверждает это: $5 \cdot 8 \equiv 1 \pmod{13}$).

Как следствие предложения 21.3 получаем такое утверждение.

Предложение 22.1. *Пусть расширенный алгоритм Евклида основывается на некоторых алгоритмах деления с остатком и умножения, битовые затраты каждого из которых применительно к целым v и w допускают оценку $O(\log v \log w)$. Тогда битовая сложность обращения в поле $\mathbb{Z}_p$ с помощью расширенного алгоритма Евклида допускает оценку $O(\log^2 p)$ при использовании числа p в качестве размера входа и, соответственно, оценку $O(m^2)$ при использовании в этом качестве битовой длины m числа p.*

Несомненным достоинством алгоритма обращения, основанного на расширенном алгоритме Евклида, состоит в том, что с его помощью обращение возможно всякий раз, когда обращаемое число и модуль взаимно просты, а не только тогда, когда модуль — простое число. Например, можно определить a такое, что $5a \equiv 1 \pmod 6$ — среди чисел, входящих в $\mathbb{I}_6$, значением a является 5. В этом смысле применимость этого алгоритма шире, чем, например, алгоритма, основанного на малой теореме Ферма.

Малая теорема Ферма. *Пусть p — простое число, a — произвольное целое. Тогда $a^p \equiv a \pmod p$.*

(Путь доказательства показан в задаче 108.) Если a не делится на p, то согласно этой теореме a^{p-2} является обратным к a по простому модулю p. Но это, вообще говоря, неверно для составных p. Например, неверно, что $5^5 \equiv 1 \pmod 6$.

С помощью расширенного алгоритма Евклида возможно обращение по модулю k любого числа a, взаимно простого с k; если $a \in \mathbb{I}_k$ или $a \in \mathbb{S}_k$, то это обращение имеет битовую сложность $O(\log^2 k)$ или $O(m^2)$, коль скоро в качестве размера входа используется $m = \lambda(k)$.

Пример 22.2. Уже говорилось, что вопрос о возможности распознавания простоты со сложностью $O(m^d)$ с некоторым $d \in \mathbb{N}$ представляет большой интерес и что, скажем, алгоритм пробных делений (примеры 1.2, 4.1, 5.2) не обладает указанным свойством даже при рассмотрении не битовой сложности, а сложности по числу делений, причем как в худшем случае, так и в среднем.

Если для некоторого $a \in \mathbb{Z}$, не делящегося на n, мы обнаружим, что не выполняется сравнение

$$a^n \equiv a \pmod n, \tag{22.2}$$

то по малой теореме Ферма из этого можно заключить, что n не является простым. Если фиксировано число a такое, что $1 < a < n$, то использование бинарного алгоритма возведения в степень с заменой результата каждого из умножений остатком от деления на n позволяет вычислить a^n и проверить выполнение сравнения (22.2), затратив $O(\log^3 n)$, или, что то же самое, $O(m^3)$ битовых операций, где $m = \lambda(n)$. Однако это не позволяет утверждать, что мы можем на основе этого распознавать простоту n за $O(m^3)$ битовых операций, так как существует бесконечно много составных n таких, что (22.2) выполняется для любых a, взаимно простых с n. Это так называемые числа Кармайкла (наименьшее из которых равно $561 = 3 \cdot 11 \cdot 17$).

Напомним, что алгоритм со сложностью $O(m^d)$, где m — размер входа, $d \in \mathbb{N}$, называют алгоритмом с полиномиально ограниченной сложностью или полиномиальным. В задаче распознавания простоты числа за размер входа берется количество m двоичных цифр числа n или же $\log_2 n$. После долгих поисков, в которых принимали участие разные исследователи, алгоритм с полиномиально ограниченной битовой сложностью был в 2002 г. предложен тремя индийскими математиками — М. Агравалом, Н. Кайалом и Н. Саксеной [1]. Этот алгоритм получил в литературе название AKS по первым буквам фамилий авторов. Мы обсудим лишь главную идею алгоритма AKS, основанием которого служит следующее необходимое и достаточное условие простоты числа n.

Предложение 22.2. *Пусть $a \in \mathbb{Z}$, $n \in \mathbb{N}$ взаимно просты, тогда n является простым, если и только если выполнено полиномиальное сравнение*

$$(x - a)^n \equiv x^n - a \pmod{n}, \tag{22.3}$$

которое надо понимать так, что числовые коэффициенты при одинаковых степенях x в полиномах, расположенных в левой и правой частях, сравнимы по модулю n.

Доказательство. Пусть n — простое. Если $n = 2$, то выполнение (22.3) проверяется непосредственно, и остается рассмотреть случай нечетного n. Коэффициенты при x^k в левой и правой частях для случая $1 < k < n$ равны соответственно $(-1)^k \binom{n}{k} a^{n-k}$ и 0. При этом $\binom{n}{k}$ делится на n, так как n простое (см. задачу 107). Коэффициенты при x^n в обеих частях (22.3) равны 1, свободные члены соответствен-

[1] См. [42].

но равны $(-a)^n$ и $-a$. В силу нечетности n достаточно показать, что $a^n \equiv a \pmod n$, а это сравнение выполняется в силу малой теоремы Ферма. Мы доказали, что если n — простое, то выполнено (22.3).

Пусть теперь n — составное. Пусть, далее, простое q и $l \in \mathbb{N}^+$ таковы, что $q^l \mid n$ и при этом неверно, что $q^{l+1} \mid n$. Имеем

$$\binom{n}{q} = \frac{(n-q+1)(n-q+2)\ldots n}{q!}.$$

Так как $q \mid n$, то произведение $(n-q+1)(n-q+2)\ldots(n-1)$ не делится на q; при этом один из входящих в n множителей, равных q, сократится со знаменателем. Поэтому $\binom{n}{q}$ не делится на q^l. Помимо этого, q^l взаимно просто с a^{n-q}, так как a и q взаимно просты. Поэтому коэффициент при x^q в левой части (22.3), равный $(-1)^{n-q}a^{n-q}\binom{n}{q}$, не делится на q^l, следовательно, не делится на n и поэтому не сравним с 0 по модулю n. Мы доказали, что если n — составное, то сравнение (22.3) не имеет места. □

Из предложения 22.2 следует, что для проверки простоты числа n можно взять любое a, взаимно простое с n (подходит, например, 1), и проверить (22.3). Но прямая такая проверка потребует $\Omega(n) = \Omega(2^m)$ битовых операций, так как раскрытие скобок в левой части (22.3) дает $n+1$ слагаемое. Алгоритм AKS предписывает проверку (22.3) по двум модулям

$$(x-a)^n \equiv x^n - a \pmod{x^r - 1, n}, \tag{22.4}$$

$r > 0$. Сравнение (22.4) понимается в том смысле, что все коэффициенты остатка от деления полинома $(x-a)^n - x^n + a$ на $x^r - 1$ (они будут целыми числами) делятся на n. Вычисление остатков от деления $(x-a)^n$ и x^n на $x^r - 1$ производится с помощью бинарного алгоритма возведения в степень, результат каждого умножения сразу же заменяется остатком от деления на $x^r - 1$. Но если взять произвольное $r \in \mathbb{N}^+$, то может оказаться, что (22.4) выполняется и при некоторых составных n. Число r должно подбираться специальным образом, и авторы показывают, что подходящее для целей алгоритма r всегда существует и может быть выбрано без существенных затрат так, что $r = O(m^5)$; после выбора r сравнение (22.4) надо проверить для «небольшого» числа «легко определяемых» a — количество этих a есть $O(\sqrt{r}m)$. Итоговая сложность алгоритма допускает оценку $O(m^{11})$. Заметим попутно, что в литературе по теории сложности часто используются оценки вида $\widetilde{O}(f(m))$ ($\widetilde{O}$ называется «*O мягкое*»),

за таким обозначением скрывается $O(f(m)\log^d m)$, где d — положительное число, значение которого не уточняется. Авторы показывают, что битовая сложность их алгоритма допускает оценку $\widetilde{O}(m^{21/2})$, указывая при этом, что если верны некоторые известные в теории чисел гипотезы (для которых имеются серьезные основания полагать, что они верны), то можно взять $r = O(m^2)$, и в этом случае сложность алгоритма в целом есть $\widetilde{O}(m^6)$.

На этом мы завершим разговор об алгоритме AKS, добавив лишь ко всему сказанному, что, например, в криптографии практикуются вероятностные (типа Монте-Карло) алгоритмы распознавания простоты, имеющие меньшую сложность, но не исключающие появления, хоть и с малой вероятностью, неверного ответа. В нашем курсе такого рода алгоритмы не рассматриваются [1].

§ 23. Булева арифметика

Сложность булевых вычислений как сложность по числу булевых операций может рассматриваться как алгебраическая сложность. Но работа с булевыми величинами является фактически работой с битами, и сложность по числу булевых операций в этом смысле совпадает с битовой сложностью. Таким образом, для алгоритмов булевой арифметики алгебраическая сложность по числу булевых операций и битовая сложность — это одно и то же. Аналогично дело обстоит и с пространственной булевой (= битовой) сложностью.

Пример 23.1. Пусть дан ориентированный граф $G = (V, E)$, $n = |V|$. Пусть $C = (c_{ij})$ — булева матрица смежности графа G: $c_{ij} = 1$, если и только если i-я вершина соединена ребром с j-й, $i, j = 1, 2, \ldots, n$. Требуется построить для G его матрицу *связности* $D = (d_{ij})$: $d_{ij} = 1$, если и только если $i = j$ или i-я вершина соединена путем из одного или более ребер с j-й, $i, j = 1, 2, \ldots, n$. Если рассматривать G как граф некоторого бинарного отношения на множестве из n элементов, то задачу можно интерпретировать как задачу построения транзитивно-рефлексивного замыкания данного бинарного отношения. Соответственно, граф, для которого D является матрицей смежности, называют транзитивно-рефлексивным замыканием графа G, а саму матрицу D — транзитивно-рефлексивным замыканием матрицы C. Для матрицы D используется обозначение C^*. На рис. 20 показан ориентированный граф и его транзитивно-рефлексивное замыкание. Соот-

[1] По поводу этих алгоритмов распознавания простоты числа см. [21, гл. 33] и [8, гл. 4].

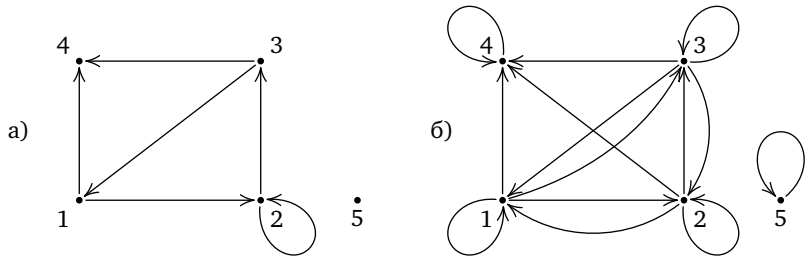


Рис. 20. а) Ориентированный граф; б) его транзитивно-рефлексивное замыкание.

ветствующие матрицы C и C^* выглядят следующим образом

$$C = \begin{pmatrix} 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}, \quad C^* = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

Нам понадобятся произведения булевых матриц, которые определяются как обычно: например, произведение F квадратных матриц A и B порядка n определяется формулой

$$f_{ij} = \bigvee_{k=1}^{n} (a_{ik} \wedge b_{kj}), \quad i, j = 1, 2, \ldots, n.$$

Элемент с индексами i, j для краткости будем называть (i, j)-элементом матрицы.

Легко видеть, что $C^2 = C \wedge C$ — это матрица, для которой (i, j)-элемент равен 1, если и только если i-я вершина соединена с j-й путем длины 2 (т. е. состоящим из двух ребер). Аналогично, C^k — матрица, для которой (i, j)-элемент равен 1, если и только если i-я вершина соединена с j-й путем длины k. Пусть I — матрица, для которой (i, j)-элемент равен 1, если и только если $i = j$. Тогда

$$C^* = I \vee C \vee C^2 \vee \ldots \vee C^{n-1}.$$

Индукцией по k несложно доказать, что для любой квадратной булевой матрицы C и натурального k выполнено

$$I \vee C \vee C^2 \vee \ldots \vee C^k = (I \vee C)^k. \tag{23.1}$$

Это дает нам

$$C^* = (I \vee C)^{n-1}. \tag{23.2}$$

Тривиальный способ умножения булевых $(n \times n)$-матриц требует $\Theta(n^3)$ битовых операций. Если используется именно этот способ, то оценка числа операций для формулы (23.2) есть $\Theta(n^4)$. Применение бинарного алгоритма (пример 4.2) для возведения в степень булевой матрицы C позволяет перейти к оценке

$$\Theta(n^3 \log n). \tag{23.3}$$

Если нежелательно связывать основанный на формуле (23.2) алгоритм построения транзитивно-рефлексивного замыкания с конкретным способом умножения матриц, то верхнюю оценку числа битовых операций можно записать как

$$n + B(n)(\lambda(n) + \lambda^*(n) - 2), \tag{23.4}$$

где $B(n)$ — сложность используемого алгоритма умножения булевых $(n \times n)$-матриц; слагаемое n отражает расход на сложение с матрицей I.

Рассмотренный пример высвечивает связь задачи построения транзитивно-рефлексивного замыкания ориентированного графа с задачей умножения булевых матриц. Эта связь оказывается очень тесной, о чем мы еще будем говорить в § 28.

В следующем примере задача построения транзитивно-рефлексивного замыкания решается без использования умножения булевых матриц.

Пример 23.2. Исследуем сложность алгоритма С. Уоршелла, предназначенного для построения матрицы C^*. В процессе применения этого алгоритма матрица C^* строится за $n = |V|$ шагов, после k-го шага получается матрица $D^{(k)} = (d_{ij}^{(k)})$, и $d_{ij}^{(k)} = 1$, если и только если i-я вершина соединена таким путем с j-й, номера всех промежуточных вершин которого не выходят за пределы множества $\{1, 2, \ldots, k\}$. Вначале полагаем $D^{(0)} = I \vee C$. Затем для каждого $k = 1, 2, \ldots, n$ находим $D^{(k)}$:

$$d_{ij}^{(k)} = d_{ij}^{(k-1)} \vee (d_{ik}^{(k-1)} \wedge d_{kj}^{(k-1)}). \tag{23.5}$$

После этого $C^* = D^{(n)}$.

Формула (23.5) отражает то, что путь из i-й вершины в j-ю, все промежуточные вершины которого принадлежат $\{1, 2, \ldots, k\}$, существует, если и только если реализуется хотя бы одна из следующих возможностей:

- имеется путь из i-й вершины в j-ю, все промежуточные вершины которого принадлежат $\{1, 2, \ldots, k-1\}$;

- имеются пути из i-й вершины в k-ю и из k-й в j-ю, все промежуточные вершины которых принадлежат $\{1, 2, \ldots, k-1\}$.

На вычисление $D^{(0)}$ уходит n битовых операций, вычисление каждой из матриц $D^{(k)}$, $1 \leqslant k \leqslant n$, требует $2n^2$ операций. Итого, требуется $2n^3 + n$ операций.

Алгоритм Уоршелла вычисляет матрицу C^*, т. е. строит транзитивно-рефлексивное замыкание ориентированного графа G с n вершинами, заданного матрицей смежности, затрачивая $2n^3 + n$ битовых операций.

Алгоритм Уоршелла можно легко реализовать так, что храниться в памяти будут только $D^{(k-1)}$ и $D^{(k)}$. Но на самом деле алгоритм Уоршелла позволяет производить все вычисления, расходуя всего n^2 битов под хранение матричных элементов. Если мы вычислим $D = I \vee C$, а затем n раз обновим матрицу D (каждый раз всю целиком, не оставляя незатронутых элементов), используя на k-м этапе обновления матрицы D присваивания

$$d_{ij} := d_{ij} \vee (d_{ik} \wedge d_{kj}),$$

$k = 1, 2, \ldots n$, то получим искомую матрицу связности. В самом деле, индукцией по k можно доказать, что после k-го этапа обновления получаем матрицу, равную $D^{(k)}$, при этом индуктивный переход от $k-1$ к k основывается на том, что имеют место очевидные соотношения

$$d_{ik}^{(k)} = d_{ik}^{(k-1)}, \quad d_{kj}^{(k)} = d_{kj}^{(k-1)},$$

$k = 1, 2, \ldots, n$, т. е. при обновлении элемента d_{ij} на k-м этапе не имеет никакого значения, был ли уже обновлен какой-то из элементов d_{ik}, d_{kj} или нет.

Если строить матрицу C^* на месте исходной матрицы C, то алгоритм можно изобразить так:

```
for l = 1 to n do c_ll := 1 od;
for k = 1 to n do
   for i = 1 to n do
      for j = 1 to n do
         c_ij := c_ij ∨ (c_ik ∧ c_kj)
      od
   od
od
```

Подводя итог рассмотрению алгоритма Уоршелла, заметим, что, как и алгоритм Евклида в расширенной версии, он дает довольно

редкий пример, когда низкие временные затраты сочетаются с малым расходом памяти.

Алгоритм Уоршелла построения замыкания ориентированного графа имеет битовую временную сложность $2n^3 + n$, *где* $n = |V|$ — *число вершин графа. Пространственная сложность этого алгоритма ограничена константой.*

При рассмотрении $|V|, |E|$ в качестве двух параметров размера входа сказанное сохраняет силу, так как затраты алгоритма Уоршелла не зависят от числа ребер заданного входного графа.

Задачи

98. Исследовать битовую сложность вычисления величины $1 + 2 + \ldots + n$ последовательными сложениями. Размером входа считать битовую длину m целого числа n.

99. Для временной битовой сложности «сверхнаивного» умножения (пример 19.1) при использовании параметров m_1, m_2 верна оценка сверху $O(2^{\max\{m_1,m_2\}}\max\{m_1, m_2\})$, а оценка $\Theta(2^{\max\{m_1,m_2\}}\max\{m_1, m_2\})$ неверна.

Указание. Неверна оценка $\Omega(2^{\max\{m_1,m_2\}}\max\{m_1, m_2\})$: она противоречила бы оценке $O(2^{m_2}m_1)$, так как для любого $N > 0$ существуют $m_1, m_2 > N$ такие, что $m_1 = 2^{m_2}$.

100. Определение чисел Фибоначчи, данное в § 10, позволяет вычислить F_n, выполнив $n - 1$ сложение. Битовая сложность этого алгоритма допускает оценку $O(n^2)$ при рассмотрении n в качестве размера входа.

101. Обосновать использованный в доказательстве предложения 21.1 переход от оценки $T^{**}_{\mathrm{ND}}(m_1, m_2) = O((m_1 - m_2 + 1)(m_2 + 1))$ к $T^{**}_{\mathrm{ND}}(m_1, m_2) = O((m_1 - m_2 + 1)m_2)$.

102. Можно ли усилить предложение 21.2, заменив приведенные там оценки $O(\log a \log b)$, $O(\log^2 a)$ на $\Theta(\log a \log b)$, $\Theta(\log^2 a)$?

103. Рассмотрим $m = \lambda(n)$ в качестве размера входа алгоритма вычисления факториала (пример 20.1). Верна ли для соответствующей временной битовой сложности оценка $O(2^m m)$?

104. К числу, первоначально равному нулю, прибавляется шаг за шагом по единице до получения значения $2^n - 1$, $n \geqslant 0$. При этих сложениях потребуется $2^n - n - 1$ переносов единицы в старший разряд.

105. Пусть p — простое, a и b — произвольные целые, k — натуральное. Тогда $(a + b)^{p^k} \equiv a^{p^k} + b^{p^k} \pmod{p}$.

106. а) Аддитивная группа кольца $\mathbb{Z}_k$ является циклической, в качестве образующей этой группы может выступать любой класс, содержащий число l, взаимно простое с k.

б) Пусть p — простое, тогда мультипликативная группа поля $\mathbb{Z}_p$, т. е. группа всех ненулевых элементов по умножению, является циклической.

107. Если число p — простое, а k — неотрицательное целое, меньшее p, то $\binom{p}{k} \equiv 0 \pmod{p}$.

Указание. Предварительно показать, что p не делит $k!$.

108. Индукцией по a доказать малую теорему Ферма.

Указание. Использовать равенство $a^p = (1 + (a-1))^p$ и предыдущую задачу.

109. Имеет место следующая *китайская теорема об остатках*: пусть $k_1, k_2, \ldots, k_n$ — взаимно простые натуральные числа (модули); тогда для любых $b_1, b_2, \ldots, b_n \in \mathbb{Z}$ найдется $f \in \mathbb{N}$ такое, что $f \equiv b_i \pmod{k_i}$, $i = 1, 2, \ldots, n$. (Задача восстановления числа по остаткам обсуждалась в древних китайских математических трактатах.) Дать конструктивное доказательство этой теоремы, т. е. доказательство, содержащее в себе алгоритм построения подходящего f (каждый такой алгоритм может быть назван китайским алгоритмом).

Указание. Пусть $k = k_1 k_2 \ldots k_n$ и $g_i = k/k_i$, $i = 1, 2, \ldots, n$. При всех $i = 1, 2, \ldots, n$ числа k_i и g_i взаимно просты, поэтому расширенным алгоритмом Евклида можно определить h_i так, что $h_i g_i \equiv 1 \pmod{k_i}$ и положить $f = \sum_{j=1}^{n} b_j h_j g_j$.

110. Исходя из того, что значение полинома $f(x)$ в точке $x = a$ равно по теореме Безу остатку от деления $f(x)$ на $x - a$, установить параллелизм между интерполяционной формулой Лагранжа и формулой для значения f, данной в указании к задаче 109.

111. Битовая сложность китайского алгоритма, эскизно обрисованного в указании к задаче 109 и основывающегося на расширенном алгоритме Евклида, наивном делении и наивном умножении, допускает оценку $O(\log^2 k)$, если в качестве размера входа рассмотреть k, и оценку $O(m^2)$, если в качестве размера входа рассмотреть битовую длину m числа k.

Указание. По поводу сложности вычисления g см. пример 20.2; сложность этапа вычисления всех g_i допускает оценку $O\left(\sum_{i=1}^{n} \log g \log k_i\right)$ и т. д.

112. Верно ли, что для битовой сложности в среднем алгоритма пробных делений справедлива оценка $\Omega\left(\left(\frac{4}{3}\right)^m\right)$?

113. Битовая сложность в среднем алгоритма наивного умножения двух неотрицательных целых чисел a и b допускает оценку $\Omega(m^2)$, а тем самым — и $\Theta(m^2)$, где $m = \max\{\lambda(a), \lambda(b)\}$.

Указание. Битовые затраты при наивном умножении a на b не могут быть меньше, чем $c\lambda(a)\lambda^*(b)$. Два случая

$$\lambda(a) = m, \quad \lambda(b) \leqslant m \quad \text{и} \quad \lambda(a) \leqslant m, \quad \lambda(b) = m$$

приводят к двум вероятностным пространствам V_1, V_2, состоящим из соответствующих пар (a, b) с равномерными распределениями вероятностей. Определить значения вероятностей событий $\lambda(a) = k$ и $\lambda(b) = k$ для произвольного $0 \leqslant k \leqslant m$ для каждого из пространств V_1, V_2 и найти математические ожидания случайных величин $\xi(a, b) = \lambda(a)$, $\eta(a, b) = \lambda^*(b)$. В силу независимости ξ и η можно воспользоваться равенством $\mathsf{E}(\xi(a, b)\eta(a, b)) = (\mathsf{E}(\xi(a, b))(\mathsf{E}\eta(a, b))$.

114. Привести полное доказательство равенства (23.1).

115. Основываясь на (23.2) и бинарном алгоритме возведения в степень, построение C^* можно выполнить с пространственной сложностью $3n^2 + O(1)$.

116. Искомая матрица C^* будет вычисляться правильно и после увеличения показателя степени в правой части (23.2).

117. Воспользовавшись утверждением, содержащимся в задаче 116, заменим показатель степени в правой части (23.2) на 2^m, где m — наименьшее целое, такое что $2^m \geqslant n - 1$. Как изменится верхняя оценка (23.4) при переходе к этой новой версии алгоритма?

118. Исследовать пространственную сложность описанного в задаче 117 алгоритма.

119. Какой наибольшей временной сложностью может обладать алгоритм умножения булевых матриц, или, другими словами, какую верхнюю границу должна допускать функция $B(n)$, чтобы временные затраты описанного в задаче 117 алгоритма допускали бы оценку $O(n^3)$?

120. Можно ли первую строку приведенного в § 23 псевдокода алгоритма Уоршелла переместить в конец этого псевдокода?

Указание. Рассмотрим алгоритм в сокращенном виде, удалив эту строку из псевдокода. Влияют ли значения диагональных элементов исходной матрицы на значения внедиагональных элементов матрицы-результата?

121. Описать версию алгоритма Уоршелла для вычисления кратчайших путей между вершинами графа G, считая, что вместо матри-

цы смежности дана матрица, содержащая длины соответствующих ребер. Если какие-то вершины не соединены, то длина ребра равна ∞. Провести анализ сложности по числу операций сложения и операций нахождения минимума (раздельно) и анализ пространственной сложности. (В этой задаче мы не рассматриваем битовую сложность.)

Глава 6

Рекуррентные соотношения как средство анализа сложности алгоритмов

§ 24. Простейшие рекуррентные уравнения

Рекуррентные соотношения (уравнения и неравенства) играют существенную роль в анализе алгоритмов. Наиболее известный и простой вид таких соотношений — это линейные рекуррентные уравнения с постоянными коэффициентами.

Пример 24.1. Вернемся к задаче 104, в которой надо доказать, что если к числу, первоначально равному нулю, прибавляется шаг за шагом по единице до достижения значения $2^n - 1$, $n \geqslant 0$, то в целом потребуется $2^n - n - 1$ переносов единиц в старшие разряды. Коль скоро заранее указан ответ, для доказательства можно использовать индукцию. Но формулу $2^n - n - 1$ можно вывести, исходя лишь из условия задачи. Обозначим через $s(n)$ исследуемое количество переносов и заметим, что если прибавлением единиц уже получено число $2^{n-1} - 1$ (на это потребуется $s(n-1)$ переносов), то очередное прибавление единицы потребует $n-1$ переносов и приведет к числу 2^{n-1}, двоичная запись которого есть $10\ldots0$ (количество нулей после единицы равно $n-1$). Далее в процессе достижения числа $11\ldots1$ (n единиц) потребуется еще $s(n-1)$ переносов. Получаем рекуррентное уравнение $s(n) = 2s(n-1) + n - 1$ или

$$s(n) - 2s(n-1) = n - 1, \tag{24.1}$$

при этом $s(0) = 0$. Характеристическое уравнение [1], соответствующее рекуррентному уравнению (24.1), имеет вид $\lambda - 2 = 0$. Общее решение однородного уравнения $s(n) - 2s(n-1) = 0$ есть $c2^n$. Правую часть уравнения (24.1) можно записать в виде квазиполинома $(n-1)1^n$. Значение 1 не является корнем характеристического уравнения, по-

[1] О методе решения линейных рекуррентных уравнений с постоянными коэффициентами см. в приложении G.

этому (24.1) обладает частным решением вида $an+b$; подставляя это выражение вместо $s(n)$ в (24.1), получаем $an+b-2(a(n-1)+b)=$ $=n-1$, и, приравнивая в левой и правой частях коэффициенты при первой и нулевой степенях n, имеем $a=b=-1$. Получаем общее решение уравнения (24.1): $s(n)=c2^n-n-1$. Подбираем значение константы c так, чтобы выполнялось $s(0)=0$; для этого должно выполняться $c\cdot 2-2=0$, т. е. $c=1$. Итак, потребуется 2^n-n-1 переносов единиц в старшие разряды.

Алгоритм, который содержит прямые или косвенные (т. е. при посредстве других алгоритмов) обращения к себе, как известно, называется *рекурсивным*. Рекуррентные соотношения являются удобным средством анализа сложности рекурсивных алгоритмов.

В следующем примере обсуждается использование рекуррентных уравнений в анализе рекурсивных алгоритмов и указывается один неудачный вид рекурсии.

Пример 24.2. Рассмотрим построение множества V_n всех целых чисел, десятичная запись каждого из которых содержит только цифры 1 и 2, а сумма цифр равна заданному числу $n\geqslant 1$. Непосредственно видно, что $V_1=\{1\}$, $V_2=\{11,2\}$, $V_3=\{111,21,12\}$. В общем случае V_n является, очевидно, объединением двух непересекающихся подмножеств, в первое из которых входят все числа из V_n, оканчивающиеся цифрой 1, во второе — цифрой 2. Если вычеркнуть последнюю цифру, то при $n\geqslant 2$ первое подмножество превратится в V_{n-1}, второе — в V_{n-2}. Это наблюдение приводит к рекурсивному алгоритму V_{Rec} построения V_n: если $n=1$ или $n=2$, то результатом будет $\{1\}$ или соответственно $\{11,2\}$; иначе надо найти V_{n-1} (рекурсивное обращение к алгоритму) и приписать справа ко всем числам этого множества единицу, затем найти V_{n-2} (еще одно рекурсивное обращение) и приписать справа ко всем числам этого множества двойку, после чего построить объединение двух получившихся множеств.

Не составляет труда заметить, что алгоритм V_{Rec} избыточно сложен (о чем еще будет идти речь). Но анализ сложности часто приходится проводить и для «плохих» алгоритмов — например, чтобы показать их практическую непригодность.

Прежде чем исследовать временную сложность этого алгоритма, установим, чему равно число $v(n)$ элементов V_n. Мы видим, что $v(1)=1$, $v(2)=2$, $v(n)=v(n-1)+v(n-2)$, $n=3,4,\ldots$ Таким образом, $v(n)$ равно $(n+1)$-му числу Фибоначчи: $v(n)=F_{n+1}$. Явное выражение $v(n)$ через n можно получить, воспользовавшись формулой Бине (10.2), которая сама может быть выведена с помощью общего

метода решения линейных рекуррентных уравнений с постоянными коэффициентами.

Используя рекуррентные уравнения, мы можем определить количество $y(n)$ преобразований чисел при построении множества V_n по описанному алгоритму (каждое из этих преобразований состоит в приписывании справа к числу единицы или двойки). Мы можем также определить число $z(n)$ выполняемых операций объединения множеств. Очевидно, имеет место уравнение $y(n) = y(n-1) + y(n-2) + F_n + F_{n-1}$, которое мы перепишем в более удобном виде и добавим начальные значения:

$$y(n) - y(n-1) - y(n-2) = F_{n+1}, \tag{24.2}$$

$y(2) = y(1) = 0$. Аналогично, имеем

$$z(n) - z(n-1) - z(n-2) = 1, \tag{24.3}$$

$z(2) = z(1) = 0$.

Начнем со второго уравнения. Будем, как обычно, прибегать к обозначениям

$$\phi = \frac{1+\sqrt{5}}{2}, \quad \tilde{\phi} = \frac{1-\sqrt{5}}{2}.$$

Непосредственно видно, что -1 является частным решением нашего рекуррентного уравнения (это частное решение можно получить и общим методом: правая часть является квазиполиномом 1^n, при этом 1 не есть корень характеристического уравнения

$$\lambda^2 - \lambda - 1 = 0, \tag{24.4}$$

и, следовательно, существует частное решение вида $c1^n$, где c — полином нулевой степени, т. е. константа; подстановка в (24.3) дает $c = -1$). Корнями уравнения (24.4) служат числа ϕ и $\tilde{\phi}$, входящие в формулу Бине, и общее решение уравнения (24.3) может быть записано в виде $z(n) = C_1\phi^n + C_2\tilde{\phi}^n - 1$. Начальные условия $z(2) = z(1) = 0$ приводят нас к системе

$$\frac{1+\sqrt{5}}{2}C_1 + \frac{1-\sqrt{5}}{2}C_2 = 1,$$
$$\left(\frac{1+\sqrt{5}}{2}\right)^2 C_1 + \left(\frac{1-\sqrt{5}}{2}\right)^2 C_2 = 1.$$

Получаем

$$C_1 = \frac{\phi}{\sqrt{5}}, \quad C_2 = -\frac{\tilde{\phi}}{\sqrt{5}}. \tag{24.5}$$

Таким образом, $z(n) = F_{n+1} - 1$. Отсюда следует ряд асимптотических формул:

$$z(n) = \frac{1}{\sqrt{5}}\phi^{n+1} + O(1), \quad z(n) \sim \frac{1}{\sqrt{5}}\phi^{n+1}, \quad z(n) = \Theta(\phi^n) \tag{24.6}$$

и т. д. Обратимся теперь к (24.2). Характеристическим уравнением здесь вновь является (24.4). Поскольку $F_{n+1} = c_1\phi^n + c_2\tilde{\phi}^n$, где c_1, c_2 — полиномы нулевой степени, то правая часть уравнения (24.2) является суммой двух квазиполиномов, и для нахождения интересующего нас решения может быть привлечен общий метод, в результате чего получится формула вида $y(n) = p_1(n)\phi^n + p_2(n)\tilde{\phi}^n$, где, в силу того, что ϕ и $\tilde{\phi}$ являются корнями кратности 1 характеристического уравнения, а также того, что $c_1, c_2 \neq 0$, степени полиномов $p_1(n)$ и $p_2(n)$ равны 1. Это показывает, что $y(n) = C_1\phi^n + C_2\tilde{\phi}^n + p_1(n)\phi^n + p_2(n)\tilde{\phi}^n = q_1(n)\phi^n + q_2(n)\tilde{\phi}^n$ при соответствующем выборе полиномов $q_1(n), q_2(n)$ первой степени. Можно было бы найти эти полиномы, но даже не делая этого, а просто принимая во внимание, что степень $q_1(n)$ равна 1, получаем

$$y(n) = \Theta(n\phi^n). \tag{24.7}$$

Мы видим, что анализ сложности простейших рекурсивных алгоритмов часто приводит к линейным рекуррентным уравнениям первого или второго порядка с постоянными коэффициентами (однородным или с правыми частями в виде суммы квазиполиномов). В таких случаях оказывается возможным найти явные формулы для сложности алгоритма. Асимптотические формулы иногда удается получить непосредственно из общего решения, не прибегая к вычислению значений входящих в него произвольных постоянных.

С помощью рекуррентных уравнений мы получаем информацию о вычислительной сложности рекурсивного алгоритма. Правда, дело не всегда обстоит просто, — иногда возникают уравнения с переменными коэффициентами или нелинейные уравнения и неравенства, и об этом мы еще будем говорить. Тем не менее, сам метод исследования сложности рекурсивных алгоритмов с помощью рекуррентных уравнений выглядит очень естественно.

Задержимся на формулах (24.6), (24.7). Если бы построение множества V_n при $n > 2$ разворачивалось не по алгоритму V_{Rec}, а несколько иначе, то затрат было бы существенно меньше. Можно находить шаг за шагом множества $V_1, V_2, V_3, \ldots$ до появления интересующего нас V_n, при этом каждое V_k, $k = 3, 4, \ldots, n$, строится исходя из уже имеющихся множеств V_{k-2} и V_{k-1}. Или же можно организовать рекурсию

для вычисления пары $P_n = (V_n, V_{n-1})$ так, чтобы при вычислении P_n исходя из P_{n-1} множество V_{n-1} просто переходило бы из пары в пару. Будем использовать обозначения $y^*(n)$ и $z^*(n)$ для количества преобразований чисел и соответственно для количества объединений множеств, затрачиваемых при нахождении (V_n, V_{n-1}). При $n > 2$ имеем $y^*(n) = y^*(n-1) + F_n + F_{n-1}$. Перепишем уравнение в более удобном виде и добавим начальное значение:

$$y^*(n) - y^*(n-1) = F_{n+1}, \tag{24.8}$$

$y^*(1) = 0$. Аналогично,

$$z^*(n) - z^*(n-1) = 1, \tag{24.9}$$

$z^*(1) = 0$. Легко получаем $z^*(n) = n - 2$ для $n \geqslant 2$ и $y^*(n) = \Theta(\phi^n)$.

Причина избыточной сложности первого из рассмотренных алгоритмов ясна. Построение V_n требует вычисления V_{n-1} и V_{n-2}, а V_{n-1} в свою очередь потребует построения V_{n-2} и V_{n-3}. Таким образом, даже не прослеживая ход построений слишком далеко, мы видим, что требуются два независимых построения множества V_{n-2}, при этом построение V_{n-2} по тем же причинам будет производиться не лучшим образом.

Пример 24.3. Достаточно ясно, что при $k > 2$ рекурсия вида

$$Y_n = U(Y_{n-1}, \ldots, Y_{n-k}) \tag{24.10}$$

(например, с заданными $Y_0, Y_1, \ldots, Y_{k-1}$) заслуживает еще меньше доверия, чем при $k = 2$, коль скоро эта рекурсия предписывает независимое вычисление $Y_{n-1}, Y_{n-2}, \ldots, Y_{n-k}$; при этом не важно, являются ли Y_i числами, множествами или же какими-то другими объектами. Нахождение Y_n с помощью простейшего циклического алгоритма потребует $n - k + 1$ обращений к U. Если при рекурсивном нахождении их требуется $y(n)$, то

$$y(n) - y(n-1) - \ldots - y(n-k) = 1, \tag{24.11}$$

$y(0) = y(1) = \ldots = y(k-1) = 0$. Можно найти асимптотику $y(n)$, не решая полностью рекуррентного уравнения (24.11), и получить следующее предложение.

Предложение 24.1. *Пусть рекурсивный алгоритм организован по схеме* (24.10)*, пусть $k \geqslant 2$ и пусть для получения значения функции U необходимы значения всех ее k аргументов. Тогда количество вычислений значений этой функции при нахождении Y_n есть $\Theta(\alpha_k^n)$, и α_k удовлетворяет условию $2 - \frac{1}{k} \leqslant \alpha_k < 2$, $k = 2, 3, \ldots$*

В приложении H можно найти полное доказательство этого предложения. Оно основывается на исследовании расположения на комплексной плоскости корней алгебраических уравнений

$$\lambda^k - \lambda^{k-1} - \ldots - \lambda - 1 = 0,$$

$k = 2, 3, \ldots$ Эти уравнения являются характеристическими для рекуррентных уравнений (24.11).

§ 25. Об одном классе нелинейных рекуррентных соотношений

Одним из источников появления рекурсии в алгоритмах служит привлечение стратегии «разделяй и властвуй»: вычислительная задача с размером входа n разбивается на $s > 1$ одноименных задач, размер входа любой из которых примерно равен n/s («разделяй»); эти задачи решаются рекурсивно, т. е. с применением этой же стратегии, а затем полученные решения соединяются в решение исходной задачи («властвуй»). Для входов маленьких размеров — как правило, 0 или 1 — задачи решаются каким-то простым способом, без рекурсии. При анализе сложности таких алгоритмов возникают специфические нелинейные рекуррентные соотношения в виде уравнений и неравенств. В общем случае такие соотношения довольно трудны для решения, но для многих конкретных задач удается исследовать сложность алгоритмов сведением рекуррентных соотношений к хорошо изученным линейным рекуррентным уравнениям первого порядка с постоянными коэффициентами и квазиполиномиальными правыми частями.

Пример 25.1. Обратимся к сортировке слияниями в ее рекурсивном варианте и исследуем ее сложность $T_{\mathrm{MS}}(n)$ по числу сравнений (n — длина массива). Функция $T_{\mathrm{MS}}(n)$ подчиняется рекуррентному неравенству

$$T_{\mathrm{MS}}(n) \leqslant \begin{cases} 0, & \text{если } n = 1, \\ T_{\mathrm{MS}}\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + T_{\mathrm{MS}}\left(\left\lceil \frac{n}{2} \right\rceil\right) + n - 1, & \text{если } n > 1. \end{cases} \tag{25.1}$$

В самом деле, для любого массива длины $n > 1$ сортировка первых его $\left\lfloor \frac{n}{2} \right\rfloor$ элементов потребует не более $T_{\mathrm{MS}}\left(\left\lfloor \frac{n}{2} \right\rfloor\right)$ сравнений (так как $T_{\mathrm{MS}}\left(\left\lfloor \frac{n}{2} \right\rfloor\right)$ — это максимальное число сравнений, которое может потребоваться рассматриваемой сортировке при работе с массивом длины $\left\lfloor \frac{n}{2} \right\rfloor$); аналогично, сортировка последних $\left\lceil \frac{n}{2} \right\rceil$ элементов потре-

бует не более, чем $T_{\mathrm{MS}}\left(\left\lceil \frac{n}{2} \right\rceil\right)$ сравнений, а слияние потребует сравнений в количестве, не превосходящем $\left\lfloor \frac{n}{2} \right\rfloor + \left\lceil \frac{n}{2} \right\rceil - 1 = n - 1$. В любом, а значит, и в худшем случае потребуется не более $T_{\mathrm{MS}}\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + T_{\mathrm{MS}}\left(\left\lceil \frac{n}{2} \right\rceil\right) + n - 1$ сравнений.

Рекуррентные неравенства, схожие с неравенством (25.1), характерны для многих алгоритмов, построенных на стратегии «разделяй и властвуй». Мы прервем на некоторое время рассмотрение этого примера и обсудим общую ситуацию с неравенствами возникающего типа.

Предложение 25.1. *Пусть вещественная функция f натурального аргумента удовлетворяет неравенству*

$$f(n) \leqslant \begin{cases} u, & \text{если } n = 1, \\ vf\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + wf\left(\left\lceil \frac{n}{2} \right\rceil\right) + \varphi(n), & \text{если } n > 1, \end{cases} \tag{25.2}$$

где u, v, w — неотрицательные вещественные числа, причем $v + w \geqslant 1$, а функция φ — неотрицательная неубывающая. Тогда

$$f(n) \leqslant t(\lceil \log_2 n \rceil), \tag{25.3}$$

где

$$t(k) = \begin{cases} u, & \text{если } k = 0, \\ (v + w)t(k-1) + \varphi(2^k), & \text{если } k > 0. \end{cases} \tag{25.4}$$

Доказательство. Установим прежде всего, что вещественная функция F натурального аргумента, удовлетворяющая равенству

$$F(n) = \begin{cases} u, & \text{если } n = 1, \\ vF\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + wF\left(\left\lceil \frac{n}{2} \right\rceil\right) + \varphi(n), & \text{если } n > 1, \end{cases} \tag{25.5}$$

является неубывающей, т. е. при $n \geqslant 2$ выполнено

$$F(n) \geqslant F(n-1). \tag{25.6}$$

Проведем индукцию по n. При $n = 2$ неравенство выполнено, так как

$$F(2) = (v + w)u + \varphi(2) \geqslant u = F(1).$$

Пусть $n > 2$ и неравенство (25.6) выполнено для $2, 3, \ldots, n-1$; докажем, что тогда оно верно и для n. Воспользуемся тем, что при $n > 2$

$$n > \left\lfloor \frac{n}{2} \right\rfloor \geqslant \left\lfloor \frac{n-1}{2} \right\rfloor \geqslant 1, \quad n > \left\lceil \frac{n}{2} \right\rceil \geqslant \left\lceil \frac{n-1}{2} \right\rceil \geqslant 1.$$

Имеем

$$F(n) = vF\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + wF\left(\left\lceil \frac{n}{2} \right\rceil\right) + \varphi(n) \geqslant$$
$$\geqslant vF\left(\left\lfloor \frac{n-1}{2} \right\rfloor\right) + wF\left(\left\lceil \frac{n-1}{2} \right\rceil\right) + \varphi(n-1) = F(n-1).$$

Неравенство (25.6) доказано.

Аналогичным образом по индукции докажем, что

$$f(n) \leqslant F(n) \tag{25.7}$$

для $n \geqslant 1$. Очевидно, $f(1) \leqslant u = F(1)$. Пусть $n > 1$ и неравенство (25.7) выполнено для $1, 2, \ldots, n-1$; докажем, что тогда оно верно и для n. В силу (25.2) мы имеем при $n > 1$

$$f(n) \leqslant vf\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + wf\left(\left\lceil \frac{n}{2} \right\rceil\right) + \varphi(n), \tag{25.8}$$

и так как при $n > 1$ выполняются неравенства $\left\lfloor \frac{n}{2} \right\rfloor \leqslant n-1$ и $\left\lceil \frac{n}{2} \right\rceil \leqslant$ $\leqslant n-1$, то по предположению индукции

$$vf\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + wf\left(\left\lceil \frac{n}{2} \right\rceil\right) + \varphi(n) \leqslant vF\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + wF\left(\left\lceil \frac{n}{2} \right\rceil\right) + \varphi(n). \tag{25.9}$$

Правая часть последнего неравенства есть $F(n)$. Это доказывает (25.7).

Свойства (25.6), (25.7) дают нам

$$f(n) \leqslant F(n) \leqslant F(2^{\lceil \log_2 n \rceil}) \tag{25.10}$$

для $n \geqslant 1$. Исследуем функцию $t(k) = F(2^k)$, $k = 0, 1, \ldots$ В силу (25.5)

$$F(2^k) = \begin{cases} u, & \text{если } k = 0, \\ (v+w)F(2^{k-1}) + \varphi(2^k), & \text{если } k > 0, \end{cases}$$

и, следовательно, $t(k)$ удовлетворяет линейному рекуррентному уравнению (25.4). Отсюда и из (25.10) следует требуемое. □

Определение 25.1. Рекуррентное уравнение (25.4), включающее в себя начальное условие $t(0) = u$, мы назовем уравнением, *ассоциированным* с рекуррентным неравенством (25.2).

Если $u, v, \varphi(n)$ удовлетворяют условиям, сформулированным в предложении 25.1, то решение ассоциированного уравнения — неотрицательная функция (см. задачу 125).

Продолжение примера 25.1. Уравнением, ассоциированным с (25.1), будет

$$t(k)=\begin{cases}0, & \text{если } k=0,\\ 2t(k-1)+2^k-1, & \text{если } k>0.\end{cases} \tag{25.11}$$

Отвлекаясь временно от начального значения $t(0)=0$, мы замечаем, что правая часть в

$$t(k)-2t(k-1)=2^k-1 \tag{25.12}$$

является суммой квазиполиномов 2^k и -1; первый из них, т. е. 2^k, интересен тем, что 2 является корнем характеристического уравнения $\lambda-2=0$, и поэтому рекуррентное уравнение $t(k)-2t(k-1)=2^k$ имеет частное решение вида $p(k)2^k$, где $p(k)$ — полином первой степени. Подстановка $(p_1k+p_0)2^k$ в рекуррентное уравнение дает

$$(p_1k+p_0)2^k-(p_1(k-1)+p_0)2^k=2^k,$$

откуда получаем, что $p_1=1$, p_0 — любое. Итак, $k2^k$ является частным решением рекуррентного уравнения с правой частью 2^k. Слагаемое -1 правой части исходного уравнения можно переписать в виде $(-1)\cdot 1^k$, единица не является корнем характеристического уравнения, поэтому рекуррентное уравнение $t(k)-2t(k)=-1$ имеет в качестве частного решения некоторую константу, значение которой определяется без труда — это 1.

Таким образом, любое решение рекуррентного уравнения (25.12) может быть записано в виде

$$c2^k+k2^k+1$$

с некоторым конкретным c. Из условия $t(0)=0$ находим $c=-1$ и

$$t(k)=k2^k-2^k+1. \tag{25.13}$$

По предложению 25.1

$$\begin{aligned}T_{\mathrm{MS}}(n)&\leqslant\lceil\log_2 n\rceil 2^{\lceil\log_2 n\rceil}-2^{\lceil\log_2 n\rceil}+1=\\&=(\lceil\log_2 n\rceil-1)2^{\lceil\log_2 n\rceil}+1\leqslant\\&\leqslant\log_2 n\cdot 2^{\log_2 n+1}+1=\\&=2n\log_2 n+1.\end{aligned}$$

Мы еще раз прервем рассмотрение примера 25.1 и сформулируем следующее предложение.

Предложение 25.2. *Пусть вещественная функция f натурального аргумента удовлетворяет неравенству*

$$f(n) \geqslant \begin{cases} u, & \text{если } n = 1, \\ vf\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + wf\left(\left\lceil \frac{n}{2} \right\rceil\right) + \varphi(n), & \text{если } n > 1, \end{cases} \tag{25.14}$$

где u, v, w — неотрицательные вещественные числа, причем $v + w \geqslant 1$, а функция φ — неотрицательная неубывающая. Тогда

$$f(n) \geqslant t(\lfloor \log_2 n \rfloor), \tag{25.15}$$

где t — решение рекуррентного уравнения (25.4).

Доказательство получается заменой знака $\leqslant$ на $\geqslant$ в (25.7), (25.8) и (25.9), а также переходом от (25.10) к

$$f(n) \geqslant F(n) \geqslant F(2^{\lfloor \log_2 n \rfloor}).$$

Сохраняя введенную терминологию, рекуррентное уравнение (25.4) будем называть ассоциированным с рекуррентным неравенством (25.14).

Если же возникает соотношение со знаком $=$ вместо $\geqslant$ или $\leqslant$, то мы можем рассматривать его как систему двух рекуррентных неравенств (25.2), (25.14). Тогда получим оценки

$$t(\lfloor \log_2 n \rfloor) \leqslant f(n) \leqslant t(\lceil \log_2 n \rceil).$$

Завершим рассмотрение примера 25.1. Можно доказать (см. задачу 133), что для сложности по числу сравнений рекурсивной сортировки слияниями выполнено

$$T_{\mathrm{MS}}(n) = \begin{cases} 0, & \text{если } n = 1, \\ T_{\mathrm{MS}}\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + T_{\mathrm{MS}}\left(\left\lceil \frac{n}{2} \right\rceil\right) + n - 1, & \text{если } n > 1, \end{cases} \tag{25.16}$$

и прийти к справедливым для всех n оценкам

$$(\lfloor \log_2 n \rfloor - 1)2^{\lfloor \log_2 n \rfloor} + 1 \leqslant T_{\mathrm{MS}}(n) \leqslant (\lceil \log_2 n \rceil - 1)2^{\lceil \log_2 n \rceil} + 1. \tag{25.17}$$

Для исследования сложности $\widetilde{T}_{\mathrm{MS}}(n)$ рекурсивной сортировки слияниями по числу перемещений рассмотрим рекуррентное уравнение

$$\widetilde{T}_{\mathrm{MS}}(n) = \begin{cases} 0, & \text{если } n = 1, \\ \widetilde{T}_{\mathrm{MS}}\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + \widetilde{T}_{\mathrm{MS}}\left(\left\lceil \frac{n}{2} \right\rceil\right) + n, & \text{если } n > 1 \end{cases} \tag{25.18}$$

(при слиянии массивов, содержащих соответственно $\left\lfloor \frac{n}{2} \right\rfloor$ и $\left\lceil \frac{n}{2} \right\rceil$ элементов, требуется ровно n перемещений элементов). Решением ассоциированного уравнения

$$t(k) = \begin{cases} 0, & \text{если } k = 0, \\ 2t(k-1) + 2^k, & \text{если } k > 0, \end{cases}$$

является $k2^k$, что дает нам

$$\lfloor \log_2 n \rfloor 2^{\lfloor \log_2 n \rfloor} \leqslant \widetilde{T}_{\mathrm{MS}}(n) \leqslant \lceil \log_2 n \rceil 2^{\lceil \log_2 n \rceil}. \tag{25.19}$$

Пространственная сложность этой сортировки есть $\Omega(n)$, ибо слияние упорядоченных массивов выполняется с получением результата на новом месте. (Дополнительно к этому будет использоваться стек отложенных заданий.) Рассмотрение примера 25.1 закончено.

Интересно посмотреть на результаты применения разобранного метода на каком-нибудь примере, для которого мы знаем точное значение сложности алгоритма, — сколь далеки будут получаемые оценки от известного точного значения?

Пример 25.2. Вернемся к бинарному алгоритму вычисления a^n (пример 4.2), который можно легко описать рекурсивно. Для его сложности по числу умножений имеем

$$T_{\mathrm{RS}}(n) \leqslant \begin{cases} 0, & \text{если } n = 1, \\ T_{\mathrm{RS}}\left(\left\lfloor \frac{n}{2} \right\rfloor \right) + 2, & \text{если } n > 1, \end{cases} \tag{25.20}$$

и

$$T_{\mathrm{RS}}(n) \geqslant \begin{cases} 0, & \text{если } n = 1, \\ T_{\mathrm{RS}}\left(\left\lfloor \frac{n}{2} \right\rfloor \right) + 1, & \text{если } n > 1. \end{cases} \tag{25.21}$$

Соответствующими ассоциированными уравнениями будут

$$t(k) = \begin{cases} 0, & \text{если } k = 0, \\ t(k-1) + 2, & \text{если } k > 1, \end{cases}$$

и

$$t(k) = \begin{cases} 0, & \text{если } k = 0, \\ t(k-1) + 1, & \text{если } k > 1, \end{cases}$$

их решения суть $2k$ и k. Отсюда

$$\lfloor \log_2 n \rfloor \leqslant T_{\mathrm{RS}}(n) \leqslant 2\lceil \log_2 n \rceil. \tag{25.22}$$

Эти оценки не сильно расходятся с точной формулой $T_{\mathrm{RS}}(n) = \lambda(n) + \lambda^*(n) - 2$.

§ 26. Асимптотические оценки решений рекуррентных неравенств

Рассмотренный метод оценивания решений рекуррентных неравенств (25.2), (25.14) приводит к достаточно точным оценкам, но требует многоэтапной работы. Часто бывает достаточно получить описание роста решения в виде простой асимптотической оценки. В § 24 мы получили оценку (24.7), указав при этом, что для ее вывода не нужно определять численные значения коэффициентов, входящих в соответствующее частное решение ассоциированного рекуррентного уравнения. Этот путь приводит к ряду общих теорем. Для них в разных литературных источниках используются синонимичные названия «теорема о рекуррентном неравенстве», «основная теорема о рекуррентных оценках» и т. д.[1] Мы приведем две теоремы такого рода.

Теорема 26.1 (о рекуррентном неравенстве, случай $\leqslant$). *Пусть неотрицательная вещественная функция f натурального аргумента удовлетворяет неравенству*

$$f(n) \leqslant \begin{cases} u, & \text{если } n = 1, \\ vf\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + wf\left(\left\lceil \frac{n}{2} \right\rceil\right) + cn^d, & \text{если } n > 1, \end{cases} \tag{26.1}$$

где u, v, w, d — неотрицательные вещественные числа, причем $v + w \geqslant 1$; c — положительное вещественное число. Тогда

$$f(n) = \begin{cases} O(n^d \log n), & \text{если } d = \log_2(v+w), \\ O(n^d), & \text{если } d > \log_2(v+w), \\ O(n^{\log_2(v+w)}), & \text{если } d < \log_2(v+w). \end{cases} \tag{26.2}$$

Доказательство. Решение ассоциированного уравнения

$$t(k) = \begin{cases} u, & \text{если } k = 0, \\ (v+w)t(k-1) + c(2^d)^k, & \text{если } k > 0, \end{cases}$$

имеет вид

$$t(k) = c_0(v+w)^k + (c_1 k + c_2)(2^d)^k = c_0(2^{\log_2(v+w)})^k + (c_1 k + c_2)(2^d)^k \tag{26.3}$$

с некоторыми конкретными c_0, c_1, c_2, причем $c_1 \neq 0$, если и только если $2^d = v + w$, или, что то же самое, если $d = \log_2(v+w)$. Рассмотрим раздельно три случая.

[1] В литературе на английском языке — «master theorem» (мастер-теорема).

Случай $d = \log_2(v+w)$. Имеем $t(k) = (c_1 k + (c_0 + c_2))(2^d)^k$. Для достаточно больших значений k выполняется

$$t(k) < Ck(2^d)^k = Ck(2^k)^d,$$

где C — некоторое положительное число. Используя предложение 25.1, заключаем, что

$$f(n) < C\lceil \log_2 n \rceil (2^{\lceil \log_2 n \rceil})^d.$$

Это дает $f(n) = O(n^d \log n)$.

Случай $d > \log_2(v+w)$. Имеем $t(k) = c_0(v+w)^k + c_2(2^d)^k$. Для достаточно больших значений k будем иметь

$$t(k) < C(2^d)^k = C(2^k)^d,$$

где C — некоторое положительное число. Аналогично предыдущему случаю с помощью предложения 25.1 получаем $f(n) = O(n^d)$.

Случай $d < \log_2(v+w)$. Вновь имеем $t(k) = c_0(v+w)^k + c_2(2^d)^k$, но теперь для достаточно больших значений k будем иметь

$$t(k) < C(v+w)^k = C(2^{\log_2(v+w)})^k = C(2^k)^{\log_2(v+w)},$$

где C — некоторое положительное число. С помощью предложения 25.1 приходим к оценке $f(n) = O(n^{\log_2(v+w)})$. □

Похожая теорема может быть доказана и для случая неравенства со знаком $\geqslant$ вместо знака $\leqslant$.

Теорема 26.2 (о рекуррентном неравенстве, случай $\geqslant$). *Пусть вещественная функция $f(n)$ натурального аргумента удовлетворяет неравенству*

$$f(n) \geqslant \begin{cases} u, & \text{если } n = 1, \\ vf\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + wf\left(\left\lceil \frac{n}{2} \right\rceil\right) + cn^d, & \text{если } n > 1, \end{cases}$$

где u, v, w, d — неотрицательные вещественные числа, причем $v + w \geqslant 1$; c — положительное вещественное число. Тогда

$$f(n) = \begin{cases} \Omega(n^d \log n), & \text{если } d = \log_2(v+w), \\ \Omega(n^{\log_2(v+w)}), & \text{если } d > \log_2(v+w), \\ \Omega(n^d), & \text{если } d < \log_2(v+w). \end{cases} \tag{26.4}$$

Доказательство отличается от доказательства теоремы 26.1 лишь тем, что в случаях $d \neq \log_2(v+w)$ вместо выбора в (26.3) слагаемого, содержащего степень двойки с бо̀льшим показателем, мы выбираем

то слагаемое, которое содержит степень двойки с меньшим показателем. Вместо предложения 25.1 пользуемся предложением 25.2. □

Справедливость следующего предложения проверяется непосредственно.

Предложение 26.1. *Если в условиях теорем 26.1 и 26.2 предположить, что* $c = 0$, *то получим соответственно* $f(n) = O(n^{\log_2(v+w)})$ *и* $f(n) = \Omega(n^{\log_2(v+w)})$.

Пример 26.1. Вновь рассмотрим сложности рекурсивной сортировки слияниями и бинарного возведения в степень. Уравнение (25.18) представим как систему двух неравенств со знаками соответственно $\leqslant$ и $\geqslant$. Применяя к ним теоремы 26.1 и 26.2 ($v + w = 2$, $\log_2(v + w) = d = 1$), из первой теоремы получаем $\widetilde{T}_{\mathrm{MS}}(n) = O(n \log n)$, из второй — $\widetilde{T}_{\mathrm{MS}}(n) = \Omega(n \log n)$. В итоге имеем $\widetilde{T}_{\mathrm{MS}}(n) = \Theta(n \log n)$. Аналогичным образом представим в виде системы двух неравенств уравнение (25.16). Заменим в правой части $n - 1$ на n в случае $\leqslant$ и на $\frac{n}{2}$ в случае $\geqslant$. Очевидно, что полученные неравенства являются следствиями исходных. С помощью теорем 26.1 и 26.2 (вновь $v + w = 2$, $\log_2(v + w) = d = 1$) получаем $T_{\mathrm{MS}}(n) = O(n \log n)$ и $T_{\mathrm{MS}}(n) = \Omega(n \log n)$. В итоге имеем $T_{\mathrm{MS}}(n) = \Theta(n \log n)$.

Сложность рекурсивной сортировки слияниями как по числу сравнений, так и по числу перемещений элементов массива допускает оценку $\Theta(n \log n)$.

Применяя теоремы 26.1 и 26.2 к (25.20), (25.21) (теперь $v + w = 1$, $\log_2(v + w) = d = 0$), получим $T_{\mathrm{RS}}(n) = O(\log n)$ и $T_{\mathrm{RS}}(n) = \Omega(\log n)$. В итоге получаем $T_{\mathrm{RS}}(n) = \Theta(\log n)$.

Асимптотические оценки $T_{\mathrm{MS}}(n) = \Theta(n \log n)$, $\widetilde{T}_{\mathrm{MS}}(n) = \Theta(n \log n)$, $T_{\mathrm{RS}}(n) = \Theta(\log n)$ можно было бы получить и из выведенных ранее неравенств (25.19), (25.17), (25.22). Этот пример демонстрирует лишь то, что теоремы о рекуррентных неравенствах позволяют быстро получать асимптотические оценки непосредственно из первоначальных рекуррентных неравенств.

Пример 26.2. Известен алгоритм построения выпуклой оболочки объединения двух выпуклых многоугольников, имеющих соответственно n_1 и n_2 вершин, со сложностью $O(n_1 + n_2)$ по общему числу операций, $n_1 + n_2$ рассматривается как размер входа этого алгоритма (см. задачу 62). Стратегия «разделяй и властвуй» позволяет, исходя из этого, описать алгоритм построения выпуклой оболочки данного

множества n точек, сложность которого по общему числу операций при выборе n в качестве размера входа определяется соотношением

$$T(n)=\begin{cases}0, & \text{если } n=1,\\ T\left(\left\lfloor\frac{n}{2}\right\rfloor\right)+T\left(\left\lceil\frac{n}{2}\right\rceil\right)+O(n) & \text{при } n\to\infty.\end{cases}$$

От этого соотношения мы переходим к неравенству

$$T(n)\leqslant\begin{cases}0, & \text{если } n=1,\\ T\left(\left\lfloor\frac{n}{2}\right\rfloor\right)+T\left(\left\lceil\frac{n}{2}\right\rceil\right)+cn, & \text{если } n>1,\end{cases}$$

где c — некоторое неизвестное нам положительное число, — мы пользуемся здесь тем, что если $g(n)=O(h(n))$ и $h(n)>0$ при $n>1$, то $g(n)\leqslant ch(n)$ для некоторой константы c и всех $n>1$.

Применяем теорему 26.1 ($v+w=2, \log_2(v+w)=1, d=1$). Это дает $T(n)=O(n\log n)$. Мы имеем, таким образом, еще один алгоритм построения выпуклой оболочки n заданных точек, по общему числу операций имеющий сложность $O(n\log n)$.

Теоремы о рекуррентных неравенствах в некоторых книгах формулируется иначе (см. задачу 144). Иногда [1] в условии этой теоремы предполагается, что исследуемая сложность является неубывающей функцией от n. Перед тем как применять теорему в таком ее виде к сложности конкретного алгоритма, необходимо доказывать неубывание этой сложности. Неубывание не является самоочевидным фактом для алгоритмов, построенных по стратегии «разделяй и властвуй». Например, для бинарного алгоритма вычисления a^n это не так: $T_{\mathrm{RS}}(7)=4, T_{\mathrm{RS}}(8)=3$. В теоремах 26.1 и 26.2 неубывание не предполагается.

Еще одно замечание. В § 9 мы получали формулы и оценки для сложностей алгоритма бинарного поиска места элемента, сортировки фон Неймана и т. д. путем сравнения возникающих величин с последовательностью $2^l, l=0,1,\ldots$; этот прием во многом аналогичен использованию соотношений вида (25.2), (25.14), но в том лишь частном случае, когда одно из чисел v, w равно нулю.

§ 27. Добавление нулей

Ряд алгоритмов целочисленной арифметики и теории матриц оказывается достаточным (без нанесения сколь-либо существенного ущерба идее алгоритма и его качеству) описать для случая, когда размер

[1] См., например, [4].

входа есть число вида 2^k. При использовании стратегии «разделяй и властвуй» это облегчает и описание алгоритма, и анализ его сложности. С теоретической точки зрения для задачи умножения двух целых чисел a и b мы можем предполагать, что битовая длина каждого из данных чисел равна 2^k, где

$$k = \max\{\lceil \log_2 \lambda(a) \rceil, \lceil \log_2 \lambda(b) \rceil\}, \tag{27.1}$$

так как всегда возможно добавить спереди любого из данных чисел некоторое количество нулей. Если речь идет об умножении квадратных матриц A и B произвольного порядка n, то мы можем добавить к матрицам несколько нулевых строк и столбцов так, чтобы сделать их порядки равными $2^{\lceil \log_2 n \rceil}$:

$$\begin{pmatrix} A & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} B & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} AB & 0 \\ 0 & 0 \end{pmatrix} \tag{27.2}$$

(нулями обозначены нулевые блоки соответствующего размера). Несмотря на переход от начальных данных к более громоздким, некоторые из алгоритмов, основанных на стратегии «разделяй и властвуй» и использующих этот переход, имеют существенно меньшую сложность, чем наивные алгоритмы.

Предложение 27.1. *Пусть вещественная функция f натурального аргумента такова, что $f(n) = f(2^{\lceil \log_2 n \rceil})$ для всех $n \in \mathbb{N}^+$ и*

$$f(2^k) \leqslant \begin{cases} u, & \text{если } k = 0, \\ wf(2^{k-1}) + \varphi(2^k), & \text{если } k > 0, \end{cases}$$

при $k \in \mathbb{N}$, где u, w — вещественные числа, причем $u \geqslant 0$, $w \geqslant 1$, а φ — неотрицательная функция, определенная для всех $n \in \mathbb{N}^+$. Тогда при всех $n \in \mathbb{N}^+$ выполняется неравенство

$$f(n) \leqslant \begin{cases} u, & \text{если } n = 1, \\ wf\left(\left\lceil \frac{n}{2} \right\rceil\right) + \varphi(2^{\lceil \log_2 n \rceil}), & \text{если } n > 1. \end{cases} \tag{27.3}$$

Доказательство. Легко видеть, что

$$\left\lceil \log_2 \left\lceil \frac{n}{2} \right\rceil \right\rceil = \lceil \log_2 n \rceil - 1. \tag{27.4}$$

В самом деле, если $2^{k-1} < n \leqslant 2^k$, $k > 1$, то $2^{k-2} < \left\lceil \frac{n}{2} \right\rceil \leqslant 2^{k-1}$, т. е. если $\lceil \log_2 n \rceil = k$, то $\left\lceil \log_2 \left\lceil \frac{n}{2} \right\rceil \right\rceil = k - 1$. Для случая $n = 2^{\lceil \log_2 n \rceil}$ неравенство (27.3) выполнено по условию, для остальных случаев используем равенства $f(n) = f(2^{\lceil \log_2 n \rceil})$, $f\left(\left\lceil \frac{n}{2} \right\rceil\right) = f(2^{\lceil \log_2 \lceil n/2 \rceil \rceil}) = f(2^{\lceil \log_2 n \rceil - 1})$. □

Теорема 27.1. *Пусть вещественная функция f натурального аргумента такова, что $f(n) = f(2^{\lceil \log_2 n \rceil})$ для всех $n \in \mathbb{N}^+$ и*

$$f(2^k) \leqslant \begin{cases} u, & \text{если } k = 0, \\ wf(2^{k-1}) + c(2^k)^d, & \text{если } k > 0, \end{cases}$$

при $k \in \mathbb{N}$, где $u, d \geqslant 0$, $c > 0$, $w \geqslant 1$. Тогда при рассмотрении f как функции, определенной для всех $n \in \mathbb{N}^+$, выполняются оценки

$$f(n) = \begin{cases} O(n^d \log n), & \text{если } d = \log_2 w, \\ O(n^d), & \text{если } d > \log_2 w, \\ O(n^{\log_2 w}), & \text{если } d < \log_2 w. \end{cases}$$

Доказательство следует из предложения 27.1 и теоремы 26.1. □

Подобно тому, как доказательство теоремы 26.1 было преобразовано в доказательство теоремы 26.2, из приведенного выше доказательства мы можем получить доказательство следующей теоремы

Теорема 27.2. *Пусть вещественная функция f натурального аргумента такова, что $f(n) = f(2^{\lceil \log_2 n \rceil})$ для всех $n \in \mathbb{N}^+$ и*

$$f(2^k) \geqslant \begin{cases} u, & \text{если } k = 0, \\ wf(2^{k-1}) + c(2^k)^d, & \text{если } k > 0, \end{cases}$$

где $u, d \geqslant 0$, $c > 0$, $w \geqslant 1$. Тогда для функции $f(n)$, при рассмотрении ее как функции, определенной для всех $n \in \mathbb{N}^+$, выполнено

$$f(n) = \begin{cases} \Omega(n^d \log n), & \text{если } d = \log_2 w, \\ \Omega(n^{\log_2 w}), & \text{если } d > \log_2 w, \\ \Omega(n^d), & \text{если } d < \log_2 w. \end{cases}$$

Перейдем к примерам.

Пример 27.1 (умножение Карацубы). Пусть a и b — целые положительные числа битовой длины $m = 2^k$. Положив $l = 2^{k-1}$, можем записать

$$a = e2^l + f, \quad b = g2^l + h,$$

где e, f, g, h — целые числа битовой длины l. А. А. Карацубе принадлежит замечательное наблюдение, позволяющее вычислить произведение ab, выполнив всего три умножения чисел половинной длины, несколько сдвигов (домножений на 2^m и 2^l) и несколько аддитивных операций над числами битовой длины $\leqslant 2m$:

$$ab = eg2^{2l} + ((e+f)(g+h) - eg - fh)2^l + fh, \tag{27.5}$$

тогда как обычное раскрытие скобок в $(e2^l+f)(g2^l+h)$ требует выполнения четырех таких умножения:

$$ab = eg2^{2l} + (eh+fg)2^l + fh. \tag{27.6}$$

Мы видим, что формула (27.5) использует произведения eg, fh, $(e+f)(g+h)$, а формула (27.6) — произведения eg, eh, fg, fh.

Небольшая проблема, которая выше была замаскирована словами «половинная длина», состоит в том, что битовая длина любого из чисел $e+f$, $g+h$, входящей в произведение $(e+f)(g+h)$, может оказаться равной $l+1$, а не l. Но если

$$e+f = e_1 2^l + f_1, \quad g+h = g_1 2^l + h_1,$$

где e_1, g_1 — однобитовые числа (0 или 1), то

$$(e+f)(g+h) = e_1 g_1 2^{2l} + (e_1 h_1 + g_1 f_1)2^l + f_1 h_1. \tag{27.7}$$

Произведение $f_1 h_1$ вычисляется рекурсивным обращением к алгоритму, произведения $e_1 g_1, e_1 h_1, g_1 f_1$, как и все сложения и сдвиги, требуют $O(l)$ операций.

Равенство (27.5) и предположение, что $m = 2^k$, приводят к рекурсивному алгоритму Карацубы умножения целых положительных чисел (будем обозначать этот алгоритм буквами КМ: первая из этих букв — начальная в фамилии автора алгоритма, вторая — в английском слове multiplication — умножение). Предположение $m = 2^k$ приводит к следующему соотношению для битовой сложности умножения Карацубы:

$$T_{\text{КМ}}(m) \leqslant \begin{cases} 1, & \text{если } m = 1, \\ 3T_{\text{КМ}}\left(\frac{m}{2}\right) + cm, & \text{если } m > 1, \end{cases} \tag{27.8}$$

где c — некоторая положительная константа.

Умножение Карацубы при произвольном входе $a, b \in \mathbb{N}^+$ размера $m = \max\{\lambda(a), \lambda(b)\}$ предполагает, что сначала мы находим $k = \lceil \log_2 m \rceil$, затем добавляем спереди каждого из a, b некоторое количество нулей так, чтобы битовая длина каждого из сомножителей стала равной 2^k, а после этого используем рекурсивный алгоритм, основанный на (27.5).

Мы можем применить теорему 27.1 ($w = 3$, $d = 1$), так как при произвольном $m \in \mathbb{N}^+$ выполняется $T_{\text{КМ}}(m) = T_{\text{КМ}}(2^{\lceil \log_2 m \rceil})$. Получаем

$$T_{\text{КМ}}(m) = O(m^{\log_2 3}), \tag{27.9}$$

при этом $\log_2 3 = 1{,}58\ldots$

Для $m > 1$ мы имеем

$$T_{\mathrm{KM}}(m) > G(m), \tag{27.10}$$

где функция G натурального аргумента такова, что $G(m) = G(2^{\lceil \log_2 m \rceil})$ для всех $m \in \mathbb{N}^+$ и

$$G(2^k) = \begin{cases} 1, & \text{если } k = 0, \\ 3G(2^{k-1}), & \text{если } k > 0, \end{cases}$$

откуда $T_{\mathrm{KM}}(m) = \Omega(m^{\log_2 3})$ по предложению 27.2. Вместе с (27.9) это дает

$$T_{\mathrm{KM}}(m) = \Theta(m^{\log_2 3}). \tag{27.11}$$

При использовании m, равного максимальной из битовых длин двух данных чисел $a, b \in \mathbb{N}^+$, в качестве размера входа битовая сложность умножения Карацубы допускает оценку

$$T_{\mathrm{KM}}(m) = \Theta(m^{\log_2 3}),$$

при том, что $T_{\mathrm{NM}} = \Theta(m^2)$ — оценка битовой сложности наивного умножения [1].

Стратегия добавления нулей особенно характерна для исследований, в которых главной целью служит преодоление некоторого сложностного барьера; последнее было и остается важным стимулом развития теории сложности.

Пример 27.2. Алгоритм Штрассена умножения двух квадратных числовых матриц A и B порядка n, являющегося степенью двойки, основан на том, что если $n = 2l$ и

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}, \quad B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix},$$

где все A_{ij}, B_{ij} — квадратные матрицы порядка l, то матрицу

$$C = AB = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix}$$

можно получить, выполнив семь умножений квадратных матриц порядка l (при том, что потребовалось бы восемь таких умножений при

[1] История создания алгоритма Карацубы и публикации о нем в 1962 г. сообщения [15] увлекательно рассказана в статье [17] самого А. А. Карацубы; особенно богат яркими историческими деталями раздел 6 этой статьи.

использовании простейшего алгоритма, основанного на определении произведения матриц):

$$\begin{aligned} X_1 &= (A_{11}+A_{22})(B_{11}+B_{22}), & X_5 &= (A_{11}+A_{12})B_{22},\\ X_2 &= (A_{21}+A_{22})B_{11}, & X_6 &= (A_{21}-A_{11})(B_{11}+B_{12}),\\ X_3 &= A_{11}(B_{12}-B_{22}), & X_7 &= (A_{12}-A_{22})(B_{21}+B_{22}),\\ X_4 &= A_{22}(B_{21}-B_{11}), \end{aligned}$$

далее используются только аддитивные операции:

$$\begin{aligned} C_{11} &= X_1+X_4-X_5+X_7, & C_{12} &= X_3+X_5,\\ C_{21} &= X_2+X_4, & C_{22} &= X_1+X_3-X_2+X_6. \end{aligned}$$

В правильности этого можно убедиться прямой проверкой.

Равенство $n=2^k$ создает возможность для рекурсивного применения алгоритма. Алгоритм Штрассена будем обозначать начальными буквами St фамилии его автора.

В приведенных формулах использовано восемнадцать сложений матриц порядка l. Сложение двух матриц порядка l требует l^2 сложений чисел. В предположении, что $n=2^k, k\in\mathbb{N}$, имеем для общего числа операций — сложений и умножений чисел:

$$T_{\mathrm{St}}(n)=\begin{cases}1, & \text{если } n=1,\\ 7T_{\mathrm{St}}\left(\frac{n}{2}\right)+18\left(\frac{n}{2}\right)^2, & \text{если } n>1.\end{cases} \tag{27.12}$$

Если $n\in\mathbb{N}^+$ произвольно, то вначале к матрицам A и B добавляются нулевые строки и столбцы (см. (27.2)) так, чтобы порядки матриц стали равными $2^{\lceil\log_2 n\rceil}$, а затем применяется описанный рекурсивный алгоритм. Рассматривая равенство (27.12) как систему двух неравенств со знаками $\leqslant$ и $\geqslant$ и применяя теоремы 27.1 и 27.2, получаем следующий результат.

Сложность $T_{\mathrm{St}}(n)$ по числу арифметических операций алгоритма Штрассена перемножения двух числовых матриц порядка n допускает оценку $T_{\mathrm{St}}(n)=\Theta(n^{\log_2 7})$, в то время как алгоритм, непосредственно следующий из определения произведения матриц, имеет сложность $\Theta(n^3)$ (при этом $\log_2 7=2{,}81\ldots$).

Что касается булевых матриц, то алгоритм Штрассена не может быть непосредственно применен для их умножения по той, например, причине, что этим алгоритмом используется вычитание, для которого нет аналога в булевой арифметике. Но матрицы A и B порядка n, состоящие из нулей и единиц, можно перемножить как

целочисленные. Каждый элемент такого произведения не превосходит n, он равен нулю, если и только если соответствующий элемент произведения булевых матриц равен нулю. Для того, чтобы в процессе применения алгоритма Штрассена к целочисленным матрицам не возникало больших промежуточных значений, здесь можно все вычисления проводить по модулю $n+1$, т. е. проводить вычисления не в кольце $\mathbb{Z}$, а в кольце $\mathbb{Z}_{n+1}$. Если $M(n)$ — некоторая верхняя граница для числа битовых операций, затрачиваемых при выполнении одной операции сложения, вычитания или умножения в $\mathbb{Z}_{n+1}$, то сложность модифицированного таким способом алгоритма Штрассена будет допускать оценку $O(n^{\log_2 7}M(n))$. Наивное умножение в $\mathbb{Z}_{n+1}$ дает $M(n)=O(\log^2 n)$. Таким образом, $M(n)$ растет очень медленно в сравнении с остальными затратами. Так как $\log_2 7 = 2{,}81\ldots$, мы можем использовать для сложности алгоритма Штрассена, модифицированного на булев случай, например, оценку $O(n^{2,82})$ или оценку $\widetilde{O}(n^{\log_2 7})$, — смысл «$O$ мягкого» объяснялся в конце § 22.

Применение алгоритма Штрассена и арифметики по модулю $n+1$ дает алгоритм умножения двух булевых матриц порядка n, битовая сложность которого допускает оценки $\widetilde{O}(n^{\log_2 7})$ и $O(n^{2,82})$.

Мы ограничились рассмотрением применения стратегии «разделяй и властвуй» для случая, когда в результате этапа разделения возникают две задачи, и каждый из размеров входа примерно вдвое меньше изначального размера. Иногда разделение приводит к трем и более задачам.

В 1963 г. А. Л. Тоом обобщил идею умножения Карацубы [1]. Пусть s — большее единицы целое. Предполагая, что битовая длина m каждого из сомножителей имеет вид s^k, $k\in\mathbb{N}$, последовательность двоичных цифр каждого из сомножителей можно разбить на s групп по s^{k-1} цифр. Тоом показал, что умножение исходных чисел сводится к $2s-1$ умножениям чисел битовой длины s^{k-1} (в умножении Карацубы $s=2$, $2s-1=3$), остальные затраты — сложения, сдвиги — будут ограничены функцией cm, где c — зависящая от выбора s константа. Здесь этап разделения приводит к $2s-1$ задачам. Для умножения Тоома (ТМ) неравенство

$$T_{\mathrm{TM}}^{(s)}(m)\leqslant\begin{cases}1, & \text{если } m=1,\\ (2s-1)T_{\mathrm{TM}}^{(s)}\left(\dfrac{m}{s}\right)+cm, & \text{если } m>1,\end{cases}\tag{27.13}$$

[1] См. [36], [4].

выполненное в случае $m = s^k$, $k \in \mathbb{N}$, и равенство

$$T^{(s)}_{\mathrm{TM}}(m) = T^{(s)}_{\mathrm{TM}}(s^{\lceil \log_s m \rceil}),$$

выполненное при произвольном $m \in \mathbb{N}^+$, приводят к оценке $T^{(s)}_{\mathrm{TM}}(m) = O(m^{\log_s(2s-1)})$ для битовой сложности алгоритма, использующего разбиение на s частей. Может быть также показано, что

$$T^{(s)}_{\mathrm{TM}}(m) = \Theta(m^{\log_s(2s-1)}). \tag{27.14}$$

Очевидно,

$$\log_s(2s-1) = \log_s 2s\left(1 - \frac{1}{2s}\right) = 1 + \log_s 2 + \log_s\left(1 - \frac{1}{2s}\right).$$

Отсюда

$$\lim_{s\to\infty} \log_s(2s-1) = 1.$$

Это означает, что для любого $\varepsilon > 0$ можно найти целое $s \geqslant 2$ такое, что умножение Тоома с разделением на s частей (битовая длина числа предполагается равной s^k, $k \in \mathbb{N}$) будет иметь битовую сложность, допускающую оценку $\Theta(m^{1+\delta})$ при некотором δ таком, что $0 < \delta \leqslant \varepsilon$; разумеется, для битовой сложности этого алгоритма справедлива оценка $O(m^{1+\varepsilon})$.

Скажем коротко об основной идее алгоритма Тоома, приводящей к неравенству (27.13). Если, как предполагалось, битовая длина каждого из сомножителей a, b есть s^k, $k \in \mathbb{N}$, то последовательность двоичных цифр каждого из сомножителей a, b можно разбить на s групп по s^{k-1} цифр:

$$a_{s-1}, \ldots, a_1, a_0; \quad b_{s-1}, \ldots, b_1, b_0.$$

Сами сомножители a, b суть значения полиномов

$$A(x) = a_{s-1}x^{s-1} + \ldots + a_1 x + a_0, \quad B(x) = b_{s-1}x^{s-1} + \ldots + b_1 x + b_0$$

в точке $x_0 = 2^{s^{k-1}}$. Полином $C(x)$, равный произведению $A(x)B(x)$, есть полином степени не выше чем $2s-2$ (мы не утверждаем, что эта степень равна $2s-2$, так как возможно, что к изначально заданным целочисленным сомножителям спереди дописывались нули), и достаточно знать значения $C(x)$ в $2s-1$ точках (узлах интерполяции) для того, чтобы затем, например, с помощью интерполяционной формулы Лагранжа найти значение

$$C(2^{s^{k-1}}), \tag{27.15}$$

равное ab. Тоом показал, что если в качестве узлов интерполяции $x_1, x_2, \ldots, x_{2s-1}$ взять числа

$$-(s-1), -(s-2), \ldots, -1, 0, 1, \ldots, s-2, s-1,$$

рекурсивно с помощью рассматриваемого алгоритма найти

$$A(x_i), \quad B(x_i), \quad C(x_i) = A(x_i)B(x_i), \quad i = 1, 2, \ldots, 2s-1,$$

и затем, пользуясь интерполяционной формулой Лагранжа, найти значение (27.15), то это приведет к (27.13), (27.14). Такое использование интерполяции заключает в себе требуемое обобщение алгоритма Карацубы.

В 1972 г. Шенхаге и Штрассен, основываясь на идее Тоома использования интерполяции полиномов в алгоритмах умножения целых чисел, получили алгоритм умножения, битовая сложность которого допускает оценку $O(m \log m \log\log m)$, мы уже упоминали этот алгоритм в § 21. Функция $m \log m \log\log m$ растет медленнее, чем $m^{1+\delta}$ при любом $\delta > 0$. Улучшение достигнуто за счет привлечения интерполяции специального вида — так называемого быстрого преобразования Фурье[1]. До настоящего времени результат Шенхаге—Штрассена остается рекордным. Шенхаге на основе этого алгоритма умножения предложил алгоритм[2] нахождения нод(a_0, a_1), сложность которого допускает оценку $O(m \log^2 m \log\log m)$, где m — битовая длина большего числа a_0 (в примере 21.2 было показано, что алгоритм Евклида имеет сложность $\Theta(m^2)$).

Необходимо сказать, что преимущества по времени выполнения рассмотренных алгоритмов перед наивным умножением и алгоритмом Евклида проявляются лишь при очень больших значениях m.

С умножением матриц положение таково, что обобщения алгоритма Штрассена в духе обобщения, предложенного Тоомом для алгоритма Карацубы, до сих пор не найдено. Используя другие идеи, Д. Копперсмит и С. Виноград в 1987 г. предложили алгоритм со сложностью $O(n^{2,376})$, где n — порядок перемножаемых квадратных матриц[3]. Этот результат остается рекордным по сей день. Существует ли для любого $\varepsilon > 0$ алгоритм умножения матриц со сложностью $O(n^{2+\varepsilon})$ — это открытый вопрос.

---

[1] Об алгоритме Шенхаге—Штрассена см., например, [5, разд. 7.5].

[2] См., например, [5, разд. 8.10].

[3] См. [47].

Задачи

122. Игра «Ханойские башни». К горизонтальной доске приделаны три вертикальных столбика. На первый нанизано n дисков, диаметры которых убывают вверх (рис. 21). Требуется, перекладывая диски

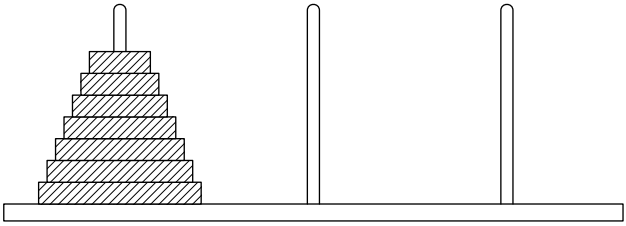


Рис. 21. Игра «Ханойские башни». Исходное положение.

по одному, расположить их в прежнем порядке на третьем столбике. Ограничение состоит в том, что больший диск никогда не может располагаться над меньшим. Разработать рекурсивный алгоритм решения этой задачи и определить его временную сложность по числу перекладываний дисков.

123. (Продолжение предыдущей задачи.) Требуется определить временную сложность алгоритма перекладывания дисков в игре «Ханойские башни» при условии, что затраты на перекладывание со столбика на столбик i-го по величине диска равна а) i; б) i^2; в) 2^i.

124. Сформулировать правило нахождения частного решения рекуррентного уравнения $a_d y(n) + a_{d-1} y(n-1) + \ldots + a_0 y(n-d) = f(n)$ для случая, когда $f(n)$ принимает постоянное значение c.

125. Пусть функция $f(n)$ определена для всех $n \in \mathbb{N}^+$ и пусть a — ненулевое число. Любое определенное для всех $n \in \mathbb{N}$ решение рекуррентного уравнения $y(n) - ay(n-1) = f(n)$ имеет вид $y(n) = a^n \left(y(0) + \sum\limits_{k=1}^{n} \frac{f(k)}{a^k} \right)$.

Указание. Индукция по n.

126. Для чисел Фибоначчи выполнено равенство $\sum\limits_{i=1}^{n} F_i = F_{n+2} - 1$, $n = 1, 2, 3, \ldots$

Указание. Решением рекуррентного уравнения $y(n) - y(n-1) = F_n$, $y(0) = 1$, является $y(n) = F_{n+2}$.

127. Верно ли, что для любого полинома $p(n)$ найдется полином $q(n)$ (оба с вещественными коэффициентами) такой, что $q(n+1) - q(n) = p(n)$, и если да, то как различаются степени полиномов $p(n)$ и $q(n)$?

128. Найти множество всех вещественных последовательностей, удовлетворяющих рекуррентному уравнению $y(n+1) + y(n) = 0$.

129. Вернемся к алгоритму V_{Rec} из § 24. Пусть $1 \leqslant k \leqslant n$. Сколько раз при построении множества V_n будет выполнено построение множества V_k?

Ответ. F_{n-k}.

130. Найти общее решение рекуррентного равенства (24.11) при $k = 1$. Найти частное решение, удовлетворяющее условию $y(0) = 0$. (При $k = 1$ рекурсия (24.10) не приводит к избыточным вычислениям значений U.)

131. Назовем перестановку $z_1, z_2, \ldots, z_n$ чисел $1, \ldots, n$ *критической* длины n, если на ней достигается максимум числа сравнений, требуемых рекурсивной сортировкой слияниями для массивов длины n. Пусть $n > 1$, а $x_1, \ldots, x_{\lfloor n/2 \rfloor}$ и $y_1, \ldots, y_{\lceil n/2 \rceil}$ суть некоторые критические перестановки длины $\left\lfloor \frac{n}{2} \right\rfloor$ и $\left\lceil \frac{n}{2} \right\rceil$. Тогда числа $z_1, z_2, \ldots, z_n$, равные, соответственно,

$$2x_1, \ldots, 2x_{\lfloor n/2 \rfloor}, 2y_1 - 1, \ldots, 2y_{\lceil n/2 \rceil} - 1$$

образуют критическую перестановку длины n (рис. 22).

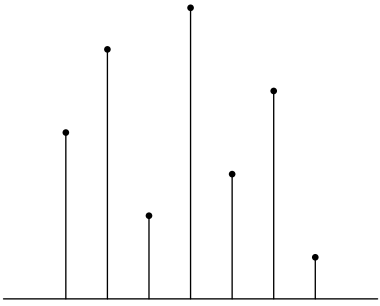


Рис. 22. Случай, когда при $n = 7$ для сортировки слияниями потребуется максимальное число сравнений.

132. Как выглядят критические перестановки длины 6, 7, 8 и 9, полученные с помощью предложенного в предыдущей задаче подхода?

133. Имеет место равенство (25.16).

Указание. См. задачу 131.

134. (Продолжение задачи 131, в которой фактически в рекурсивной форме описан алгоритм построения некоторой критической перестановки длины n.) Исследовать сложности описанного алгоритма построения некоторой критической перестановки длины n по числу умножений на два и по числу вычитаний единицы. Предложить нерекурсивный алгоритм (например, можно рассмотреть последовательное построение критических перестановок, длины которых суть соответственно $1, 2, \ldots, n$) и исследовать его сложности по названным операциям.

135. Чему равно наименьшее n вида 2^k, для которого число сравнений при рекурсивной сортировке слияниями превосходит $\lceil \log_2 n! \rceil$?

136. Оценка (25.22) является следствием оценки $T_{\mathrm{RS}}(n) = \lambda(n) + \lambda^*(n) - 2$.

137. Для сложности алгоритма бинарного поиска места элемента в упорядоченном массиве обосновать рекуррентное неравенство

$$T_{\mathrm{BS}}(n) \leqslant \begin{cases} 1, & \text{если } n = 1, \\ T_{\mathrm{BS}}\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + 1, & \text{если } n > 1, \end{cases}$$

не прибегая к известному явному выражению для $T_{\mathrm{BS}}(n)$. Из этого рекуррентного неравенства получить оценку для $T_{\mathrm{BS}}(n)$.

138. Указать алгоритм построения пересечения n полуплоскостей

$$a_i x + b_i y + c_i \geqslant 0, \quad i = 1, 2, \ldots, n,$$

имеющий временную сложность $\Theta(n \log n)$ по общему числу операций.

Указание. Пересечением будет выпуклый многоугольник (возможно, неограниченный). Опираясь на алгоритм Шеймоса—Хоя (задача 14), использовать стратегию «разделяй и властвуй».

139. По сравнению с теоремами 26.1, 26.2 предложения 25.1, 25.2 представляют собой более сильные утверждения, имеющие к тому же более широкое применение. Используя эти предложения, получить асимптотические оценки для

$$f(n) = \begin{cases} 0, & \text{если } n = 1, \\ f\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + f\left(\left\lceil \frac{n}{2} \right\rceil\right) + \varphi(n), & \text{если } n > 1, \end{cases}$$

при $\varphi(n) = \log_2 n$ и $\varphi(n) = n \log_2 n$.

140. Рекуррентное неравенство вида (26.1) может иметь более одного решения, однако во всех трех случаях, предусмотренных формулой (26.2), по крайней мере для одного решения соответствующая оценка из (26.2) является точной.

141. Обобщить предложенную в этой главе технику решения рекуррентных неравенств на случай, когда задача с размером входа n разбивается на s задач, размер каждой из которых — это $\left\lfloor \frac{n}{s} \right\rfloor$ или $\left\lceil \frac{n}{s} \right\rceil$, где s — некоторое натуральное число $\geqslant 2$. Как будут выглядеть теоремы 26.1, 26.2 в этом случае?

142. Доказать соотношение (27.10).

Указание. Рассмотреть количество умножений чисел битовой длины 1 в процессе применения алгоритма Карацубы к входу размера m и показать, что $G(m)$ является для него нижней оценкой.

143. а) Для любого $\varepsilon > 0$ можно указать $N > 0$ такое, что $3T_{\text{KM}}(m) < T_{\text{KM}}(2m) < (3+\varepsilon)T_{\text{KM}}(m)$ при всех $m \geqslant N$.

б) Для любого $\varepsilon > 0$ можно указать $N > 0$ такое, что $7T_{\text{St}}(n) < T_{\text{St}}(2n) < (7+\varepsilon)T_{\text{St}}(n)$ при всех $n \geqslant N$.

Указание. См. (27.10), (27.8) и, соответственно, (27.12).

144. Теоремы о рекуррентных неравенствах часто формулируются иначе. Например, в [4] и [5] теорема дается, с точностью до обозначений и используемых слов, в следующем виде. Пусть s — целое $\geqslant 2$, и при любом n вида s^k, $k = 1, 2, \ldots$, вещественная функция $f(n)$ натурального аргумента удовлетворяет неравенству

$$f(n) \leqslant wf\left(\frac{n}{s}\right) + cn^d,$$

где w, c, d — константы, причем $w > 0$, $c > 0$, $d \geqslant 0$. Пусть при этом $f(n)$ не убывает на каждом отрезке $[s^{k-1}+1, s^k]$. Тогда

$$f(n) = \begin{cases} O(n^d \log n), & \text{если } d = \log_s w, \\ O(n^d), & \text{если } d > \log_s w, \\ O(n^{\log_s w}), & \text{если } d < \log_s w. \end{cases}$$

Доказать эту теорему. (Заметим, что в условии теоремы 26.1 нет требования неубывания $f(n)$ на каждом отрезке $[s^k+1, s^k]$, но зато требуется, чтобы, например, неравенство (26.1) выполнялось для всех n, а не только для n вида 2^k.)

Глава 7

Сводимость

В этой главе мы рассматриваем только временную сложность алгоритмов и считаем, что размер входа — это неотрицательное целое число.

§ 28. Линейная сводимость

Ниже в сопоставляемых друг с другом вычислительных задачах подразумеваются преобразования каких-то математических объектов, представляемых одинаково для всех задач, участвующих в сопоставлении. Размер входа для алгоритмов решения рассматриваемых задач тоже должен определяться единообразно. При сравнительном исследовании задач затраты на выполнение вычислений измеряются количеством операций из одного и того же набора (например, арифметических операций, битовых операций и т. д.).

Определение 28.1. Пусть P и Q — две вычислительные задачи. Если любому алгоритму A_Q решения задачи Q можно сопоставить такой алгоритм A_P решения задачи P, что

$$T_{A_P}(n) = O(T_{A_Q}(n)), \tag{28.1}$$

то говорят, что задача P *линейно сводится* к задаче Q, и пишут $P \leqslant Q$. В специально оговариваемых случаях утверждение $P \leqslant Q$ предполагает, что рассматриваются лишь такие алгоритмы A_Q, сложности которых удовлетворяют некоторым фиксированным условиям.

Слово «линейно» в этом определении означает лишь то, что сложность получаемого алгоритма A_P не является, скажем, квадратом, кубом и т. д. сложности алгоритма A_Q. Возможное присутствие условий (ограничений) на сложности алгоритмов решения задачи Q облегчает или просто делает возможным доказательство (28.1). В то же время, предполагается, что эти ограничения отсекают нерациональные алгоритмы; тогда смысл отношения $P \leqslant Q$ состоит в том, что каж-

дому приемлемому («хорошему») алгоритму решения задачи Q мы можем сопоставить алгоритм решения задачи P такой, что выполнено (28.1).

Пример 28.1. Будем рассматривать задачи умножения M и возведения в квадрат S неотрицательных целых чисел, заданных в двоичной системе, считая размером входа в первой задаче максимальную из битовых длин чисел n_1, n_2, а во второй — битовую длину данного числа n (будем обозначать этот размер через m), в качестве затрат будем рассматривать битовые затраты.

Из равенства $n^2 = n \cdot n$ следует, что $S \leqslant M$, так как это равенство дает требуемый определением 28.1 алгоритм. Дает ли равенство

$$n_1 \cdot n_2 = \frac{(n_1+n_2)^2 - n_1^2 - n_2^2}{2} \tag{28.2}$$

основание считать, что $M \leqslant S$? Битовая длина числа $n_1 + n_2$ может оказаться равной $m+1$. Если бы для возведения в квадрат использовался некоторый чрезвычайно нерациональный алгоритм, имеющий, скажем, сложность $m!$, то соответствующий алгоритм умножения имел бы сложность порядка $(m+1)!$. Легко видеть, что равенство $(m+1)! = O(m!)$ места не имеет. Можно пытаться определить умножение через возведение в квадрат как-то иначе, но можно и не отказываться от формулы (28.2): при установлении линейной сводимости обычно считается, что сложность любого приемлемого алгоритма выполнения какой-либо мультипликативной арифметической операции (умножения, возведения в квадрат, деления и т. д.) удовлетворяет, хотя бы для всех достаточно больших m, условиям

1) $T(m) \geqslant m$,
2) $T(m)$ не убывает при возрастании m,
3) $T(2m) \leqslant 4T(m)$

(условие 3 отсекает алгоритмы умножения, сложность которых растет быстрее чем m^2). Приняв, что сложность алгоритма A_S удовлетворяет этим условиям, мы имеем

$$T_{A_S}(m+1) \leqslant T_{A_S}(2m) \leqslant 4T_{A_S}(m),$$

что дает $T_{A_M}(m) = O(T_{A_S}(m))$ для описанного с помощью (28.2) алгоритма умножения. Мы используем здесь то, что операции вычитания и деления на 2 имеют сложность $O(m)$, и то, что $T_{A_S}(m) \geqslant m$ по условию 1.

Таким образом, если принимаются ограничения 1—3 для сложностей алгоритмов решения задачи S, то $M \leqslant S$.

Пример 28.2. Из формулы (23.2) нельзя сделать вывода, что задача построения транзитивно-рефлексивного замыкания графа с n вершинами (или, в другой терминологии, транзитивно-рефлексивного замыкания булевой матрицы порядка n) линейно сводится к задаче умножения двух булевых матриц порядка n, так как само число умножений матриц в этой формуле существенно зависит от n. Но мы покажем, что соответствующая линейная сводимость имеет место, если принять, что для любого из рассматриваемых алгоритмов умножения булевых матриц порядка n его сложность $B(n)$ (примем это упрощенное обозначение) удовлетворяет, хотя бы для всех достаточно больших n, условиям

1) $B(1) = 1$,

2) $B(n)$ не убывает при возрастании n,

3) $4B(n) \leqslant B(2n) \leqslant 8B(n)$.

Комментарий к условию 3. Неравенство $B(2n) \leqslant 8B(n)$ отсекает те алгоритмы умножения булевых матриц, сложность которых растет быстрее чем n^3. Алгоритм должен построить n^2 элементов матрицы-произведения, поэтому его сложность не может расти медленнее чем n^2, и в соответствии с этим в ограничения включается неравенство $4B(n) \leqslant B(2n)$.

Для булевой матрицы X порядка n будем, как и прежде, обозначать ее транзитивно-рефлексивное замыкание через X^*.

Пусть G — ориентированный граф с n вершинами и C — его матрица смежности. Предположим вначале, что n — четное число: $n = 2l$. Разобьем множество вершин $V = \{v_1, v_2, \ldots, v_{2l}\}$ графа G на два подмножества $V_1 = \{v_1, v_2, \ldots, v_l\}$ и $V_2 = \{v_{l+1}, v_{l+2}, \ldots, v_{2l}\}$, а множество ребер графа G — на четыре подмножества $E_{11}, E_{12}, E_{21}, E_{22}$: начало каждого из ребер из множества E_{pq} принадлежит V_p, а конец — V_q, где $p, q \in \{1, 2\}$. Если исходная матрица C представлена в блочном виде

$$\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix},$$

где каждый блок — это матрица порядка l, то матрица C_{pq} несет полную информацию о ребрах из множества E_{pq}. Рассмотрим пути в графе G, соединяющие вершины из множества V_1. Назовем *ходом* путь из вершины в вершину этого множества, представляющий собой

- либо продвижение по ребру, принадлежащему E_{11},
- либо продвижение по ребру, принадлежащему E_{12}, за которым следует некоторый путь по ребрам из E_{22} и затем продвижение по ребру, принадлежащему E_{21}.

Непосредственно видно, что (i, j)-элемент матрицы $C_{11} \vee (C_{12} \wedge C_{22}^* \wedge C_{21})$ равен 1, если и только если существует ход, соединяющий вершины $v_i, v_j \in V_1$. При этом ясно, что из v_i можно добраться в v_j $(v_i, v_j \in V_1)$ по ребрам графа G, если и только если существует последовательность ходов, приводящая из v_i в v_j, т. е. если и только если (i, j)-элемент матрицы

$$(C_{11} \vee (C_{12} \wedge C_{22}^* \wedge C_{21}))^*$$

равен 1. Обозначим эту матрицу F_{11}. Мы имеем

$$C^* = \begin{pmatrix} F_{11} & F_{12} \\ F_{21} & F_{22} \end{pmatrix}$$

для некоторых матриц F_{12}, F_{21}, F_{22}, при этом матрица F_{pq}, $p, q \in \{1, 2\}$, несет информацию о существовании путей из вершин множества V_p в вершины множества V_q. Рассуждениями, сходными с приведенными выше, можно показать, что

$$\begin{aligned} F_{12} &= F_{11} \wedge C_{12} \wedge C_{22}^*, \\ F_{21} &= C_{22}^* \wedge C_{21} \wedge F_{11}, \\ F_{22} &= C_{22}^* \vee (C_{22}^* \wedge C_{21} \wedge F_{11} \wedge C_{12} \wedge C_{22}^*); \end{aligned}$$

эти выражения удобны для вычисления матрицы C^*: четыре матрицы $F_{p,q}$, $p, q \in \{1, 2\}$, можно найти, выполнив два построения транзитивно-рефлексивных замыканий, шесть умножений и два сложения матриц порядка l:

$$\begin{aligned} U &= C_{22}^*, \\ V &= C_{12} \wedge U, \\ W &= U \wedge C_{21}, \\ F_{11} &= (C_{11} \vee (V \wedge C_{21}))^*, \\ F_{12} &= F_{11} \wedge V, \\ F_{21} &= W \wedge F_{11}, \\ F_{22} &= U \vee (F_{21} \wedge V). \end{aligned}$$

Мы указали рекурсивный переход от матриц порядка $2l$ к матрицам порядка l. Учитывая, что $C^* = C$ для матрицы первого порядка, мы получаем алгоритм вычисления C^* для случаев, когда порядок n матрицы равен 2^k, $k \geqslant 0$. Пусть для умножения булевых матриц используется алгоритм со сложностью $B(n)$, которая удовлетворяет условиям 1—3, приведенным в начале этого примера. Для $n = 2^k$ мы имеем следующее соотношение, которому будет удовлетворять сложность

$T(n)$ полученного алгоритма построения транзитивно-рефлексивного замыкания:

$$T(2^k) \leqslant \begin{cases} 0, & \text{если } k = 0, \\ 2T(2^{k-1}) + 6B(2^{k-1}) + 2 \cdot 2^{2(k-1)}, & \text{если } k > 0. \end{cases} \tag{28.3}$$

В силу условия 3 имеем $B(2^{k-1}) \geqslant 4B(2^{k-2})$ при $k \geqslant 2$. Дополнительное использование условия 1 дает нам $B(2^{2k-1}) \geqslant 2^{2(k-1)}$ при $k \geqslant 1$. Заменяя $2^{2(k-1)}$ во второй строке соотношения (28.3) на $B(2^{k-1})$, получаем

$$T(2^k) \leqslant \begin{cases} 0, & \text{если } k = 0, \\ 2T(2^{k-1}) + 8B(2^{k-1}), & \text{если } k > 0. \end{cases} \tag{28.4}$$

Докажем по индукции, что

$$T(2^k) \leqslant 4B(2^k), \quad k = 0, 1, \ldots \tag{28.5}$$

Для $k = 0$ это неравенство очевидно, так как $T(1) = 0, B(1) = 1$. Пусть $k > 0$ и неравенство выполнено для $k - 1$. Тогда $2T(2^{k-1}) \leqslant 8B(2^{k-1})$, и, как следствие (28.4), мы получаем $T(2^k) \leqslant 16B(2^{k-1})$. В силу условия 3 мы имеем $B(2^{k-1}) \leqslant \frac{1}{4}B(2^k)$, откуда следует (28.5).

Если n не есть число вида 2^k, то от матрицы C переходим к матрице

$$C' = \begin{pmatrix} C & 0 \\ 0 & I_s \end{pmatrix},$$

порядка 2^k, взяв единичную матрицу I_s некоторого порядка, меньшего чем порядок матрицы C. Таким образом, порядок матрицы C' меньше, чем $2n$, т. е. $2^k < 2n$. Из (28.5) следует, что

$$T(n) = T(2^k) \leqslant 4B(2^k).$$

Так как $2^k < 2n$ и функция $B(n)$ — неубывающая, получаем $4B(2^k) \leqslant 4B(2n)$. Отсюда

$$T(n) \leqslant 4B(2n) \leqslant 4 \cdot 8 \cdot B(n) = 32B(n)$$

для всех n и

$$T(n) = O(B(n)), \tag{28.6}$$

что и требовалось.

Из сказанного следует, что если использовать для умножения квадратных булевых матриц алгоритм со сложностью, растущей медленнее, чем n^3, и удовлетворяющей условиям 1—3, то можно строить транзитивно-рефлексивное замыкание со сложностью, растущей медленнее, чем сложность алгоритма Уоршелла (пример 23.2). Напомним, что

алгоритм Штрассена умножения матриц, модифицированный для булева случая (пример 27.2), имеет сложность $\widetilde{O}(n^{\log_2 7})$. Но для небольших значений n условия 1—3 здесь выполняться не будут. Однако соотношение (28.6) можно доказать при более слабых предположениях, чем использованные выше, и обсуждавшееся доказательство не потребует больших изменений. А именно, достаточно предположить, что $B(n)$ удовлетворяет условиям 2—3 для $n > n_0$, где n_0 — некоторое натуральное число (см. задачу 143б). Получится оценка $T(n) \leqslant cB(n)$, где c зависит от $B(n_0)$, но эта зависимость не влияет на (28.6).

Для нахождения транзитивно-рефлексивного замыкания ориентированного графа с n вершинами существует алгоритм, сложность которого допускает оценку $O(n^{2,82})$, или, более точно, оценку $\widetilde{O}(n^{\log_2 7})$.

Как показывает последний пример, если $P \leqslant Q$, то прогресс в умении быстро решать задачу Q может в некоторых случаях обеспечить прогресс и в умении быстро решать задачу P. Но это не всегда так. Допустим, что n является нижней границей сложности для алгоритмов решения задачи Q и в то же время для задачи P имеется алгоритм решения со сложностью, меньшей n. Согласно определению 28.1 мы имеем $P \leqslant Q$, но наличие этой линейной сводимости ничего не дает для продвижения в умении быстро решать задачу P.

Отношение линейной сводимости, очевидно, рефлексивно. Если не рассматривать упомянутых в определении 28.1 дополнительных условий, то это отношение будет транзитивным. В тех случаях, когда такие условия наложены, необходима осмотрительность: если для задач P, Q и R мы имеем $P \leqslant Q$ при некоторых условиях на сложность алгоритмов решения Q, а также $Q \leqslant R$ при соответствующих условиях на сложность алгоритмов решения R, то чтобы на основании этого утверждать, что $P \leqslant R$, надо дополнительно установить, что сложности тех алгоритмов решения задачи Q, которые сопоставляются алгоритмам решения задачи R, удовлетворяют условиям, налагаемым на сложности алгоритмов решения задачи Q при доказательстве отношения $P \leqslant Q$. Это обстоятельство иногда безосновательно игнорируется.

§ 29. Линейная сводимость и нижние границы сложности

Нижеприведенная теорема является следствием определения 28.1.

Теорема 29.1. *Пусть задачи P и Q таковы, что $P \leqslant Q$. Пусть $f(n)$ — асимптотическая нижняя граница сложности алгоритмов решения задачи P, и при этом соотношение $P \leqslant Q$ устанавливается без*

наложения ограничений на сложность алгоритмов решения задачи Q. Тогда $f(n)$ является асимптотической нижней границей сложности алгоритмов решения задачи Q.

Доказательство. Для каждого алгоритма A_Q решения задачи Q найдется такой алгоритм A_P решения задачи P, для которого $T_{A_P}(n) = O(T_{A_Q}(n))$, или, эквивалентно, $T_{A_Q}(n) = \Omega(T_{A_P}(n))$. Но $T_{A_P}(n) = \Omega(f(n))$, следовательно, $T_{A_Q}(n) = \Omega(f(n))$. □

Если при соблюдении условия этого предложения для какого-то алгоритма A_Q решения задачи Q имеет место оценка $T_{A_Q}(n) = O(f(n))$, то этот алгоритм будет оптимальным по порядку сложности и $T_{A_Q}(n) = \Theta(f(n))$. (При наличии ограничений на сложность алгоритма A_Q, когда фактически предполагается, что A_Q принадлежит некоторому классу $\mathcal{Q}$, речь должна бы идти об оптимальности по порядку сложности в классе $\mathcal{Q}$; но если ограничения таковы, что в класс $\mathcal{Q}$ попадают наиболее рациональные алгоритмы, то упоминание класса $\mathcal{Q}$ не обязательно.)

Пример 29.1. Вновь рассмотрим задачу построения выпуклой оболочки конечного множества точек с помощью арифметических операций и сравнений. Многоугольник, являющийся выпуклой оболочкой, представляется массивом своих вершин в порядке их следования при обходе в некотором направлении, обычно против часовой стрелки, начиная с некоторой вершины. Мы покажем, что задача сортировки массивов вещественных чисел с помощью арифметических операций и сравнений линейно сводится к задаче построения выпуклой оболочки в этой постановке, считая, что в этих задачах затраты измеряются общим числом арифметических операций и сравнений. Но если порядок вершин выпуклой оболочки, которую надо построить, может быть произвольным, то задача меняется, и про нее мы ничего не утверждаем.

Пусть $x_1, x_2, \ldots, x_n$ — данный массив попарно различных вещественных чисел. Решив задачу построения выпуклой оболочки множества точек с координатами

$$(x_1, x_1^2), (x_2, x_2^2), \ldots, (x_n, x_n^2)$$

(см. рис. 23), мы можем, двигаясь по вершинам построенного многоугольника, найти вершину с наименьшей абсциссой, а затем построить массив вершин в порядке возрастания абсцисс. Сложность этой дополнительной части работы допускает оценку $O(n)$. Легко видеть, что сложность любого алгоритма построения выпуклой оболочки не

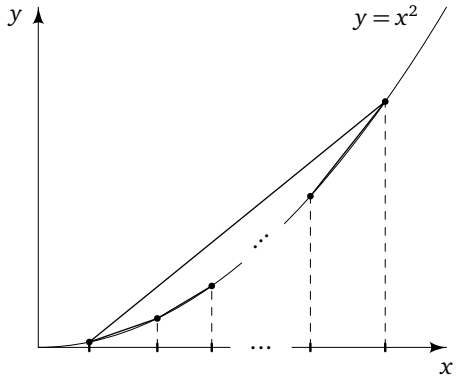


Рис. 23. Сведение сортировки чисел $x_1, x_2, \ldots, x_n$, отмеченных на оси абсцисс, к построению выпуклой оболочки точек $(x_1, x_1^2), (x_2, x_2^2), \ldots, (x_n, x_n^2)$.

может быть меньше чем n. Таким образом, каждому алгоритму A построения выпуклой оболочки мы сопоставляем алгоритм сортировки со сложностью $O(T_A(n))$. Следовательно, задача сортировки вещественных чисел с помощью арифметических операций и сравнений линейно сводится к задаче построения выпуклой оболочки.

Из результатов § 14 следует, что любая сортировка с помощью сравнений массивов длины n имеет сложность не менее $\log_2 n!$. Но при построении выпуклой оболочки используются как сравнения, так и арифметические операции. Можно ли, привлекая помимо операций сравнения еще и арифметические операции, предложить алгоритм сортировки массивов вещественных чисел, сложность которого по числу сравнений была бы меньше, чем $\log_2 n!$, пусть даже при том, что потребовалось бы очень большое число арифметических операций? Ниже мы обосновываем отрицательный ответ на этот вопрос.

Дополнительное использование четырех арифметических операций означает, что сравниваться могут не только числа $x_1, x_2, \ldots, x_n$, заданные изначально, но и значения рациональных функций от этих чисел. В качестве знаков сравнения могут использоваться

$$<,\ >,\ =,\ \neq,\ \leqslant,\ \geqslant.$$

Каждая рациональная функция $F(x_1, x_2, \ldots, x_n)$ записывается как отношение двух полиномов

$$F(x_1, x_2, \ldots, x_n) = \frac{p(x_1, x_2, \ldots, x_n)}{q(x_1, x_2, \ldots, x_n)} \tag{29.1}$$

(в этом параграфе мы рассматриваем рациональные функции и полиномы с вещественными коэффициентами). Каждый полином $f(x_1, x_2, \ldots, x_n)$ можно рассматривать как рациональную функцию

$$\frac{f(x_1, x_2, \ldots, x_n)}{1}.$$

Предполагается, что если алгоритм предписывает сравнение, в котором участвует значение рациональной функции (29.1), то ее знаменатель $q(x_1, x_2, \ldots, x_n)$ не обращается в 0 при рассматриваемых значениях $x_1, x_2, \ldots, x_n$. Неравенство $F_1(x_1, x_2, \ldots, x_n) < F_2(x_1, x_2, \ldots, x_n)$ равносильно, очевидно, неравенству $F(x_1, x_2, \ldots, x_n) < 0$, где

$$F(x_1, x_2, \ldots, x_n) = F_1(x_1, x_2, \ldots, x_n) - F_2(x_1, x_2, \ldots, x_n).$$

То же самое для сравнений со знаками $>, =, \neq, \leqslant, \geqslant$.

Далее будут использоваться два свойства рациональных функций:

(R1) Множество точек $(x_1, x_2, \ldots, x_n) \in \mathbb{R}^n$, для которых данная рациональная функция (29.1) неопределена или равна нулю, является замкнутым; в свою очередь, те точки, в которых эта функция определена и имеет значение большее (меньшее) нуля, образуют открытое множество. Это следует из того, что рациональная функция непрерывна на своем множестве определения.

(R2) Если рациональная функция (29.1) определена и равна нулю всюду на некотором непустом открытом подмножестве множества $\mathbb{R}^n$, то ее числитель $p(x_1, x_2, \ldots, x_n)$ является нулевым полиномом (см. задачу 147).

Предложение 29.1. *Функция $f(n) = \lceil \log_2 n! \rceil$ является нижней границей сложности по числу сравнений алгоритмов сортировки массивов длины n попарно различных вещественных чисел с помощью сравнений и четырех арифметических операций.*

Доказательство.[1] Каждой из перестановок $a = (a_1, a_2, \ldots, a_n) \in \Pi_n$ сопоставим *сектор* S_a — подмножество пространства $\mathbb{R}^n$ такое, что $(x_1, x_2, \ldots, x_n) \in S_a$, если и только если числа $x_1, x_2, \ldots, x_n$ попарно различны и их относительный порядок совпадает с относительным порядком чисел $a_1, a_2, \ldots, a_n$. Любой сектор является открытым множеством в $\mathbb{R}^n$. Каждому массиву вещественных чисел с попарно различными элементами $x_1, x_2, \ldots, x_n$ соответствует точка некоторого

[1] Идея элементарного доказательства сообщена автору С. П. Поляковым. Наиболее раннее (довольно трудное для понимания) доказательство было опубликовано Н. Фридманом в [49].

сектора пространства $\mathbb{R}^n$. В примере 14.2 говорилось, что при фиксированном n любую сортировку массивов длины n можно представить в виде дерева, каждой не являющейся листом вершине v которого приписано сравнение вида $x_i < x_j$, а выходящие из нее ребра имеют метки 1 («да») и 0 («нет»); при этом каждому листу l приписан набор $x_{i_1}, x_{i_2}, \ldots, x_{i_n}$, где $(i_1, i_2, \ldots, i_n)$ — некоторая перестановка чисел $1, 2, \ldots, n$.

Алгоритм сортировки массивов длины n с помощью сравнений и четырех арифметических операций представляется деревом D, которое можно считать таким, что каждой не являющейся листом вершине v приписано сравнение $F_v(x_1, x_2, \ldots, x_n)$ с нулем, где $F_v(x_1, x_2, \ldots, x_n)$ — некоторая рациональная функция, а выходящие из вершины ребра имеют метки 1 («да») и 0 («нет»). При этом каждому листу l приписаны рациональные функции

$$G_{l1}(x_1, x_2, \ldots, x_n), \quad G_{l2}(x_1, x_2, \ldots, x_n), \quad \ldots, \quad G_{ln}(x_1, x_2, \ldots, x_n), \qquad (29.2)$$

значения которых для исходных $x_1, x_2, \ldots, x_n$ определяют требуемый порядок $x_{i_1}, x_{i_2}, \ldots, x_{i_n}$:

$$\begin{aligned} x_{i_1} &= G_{l1}(x_1, x_2, \ldots, x_n), \\ x_{i_2} &= G_{l2}(x_1, x_2, \ldots, x_n), \\ &\ldots\ldots\ldots\ldots\ldots\ldots \\ x_{i_n} &= G_{ln}(x_1, x_2, \ldots, x_n). \end{aligned}$$

Можно считать, что числитель каждой из функций $F_v(x_1, x_2, \ldots, x_n)$ не является нулевым полиномом, — в противном случае вершину v можно было бы удалить из дерева вместе с выходящей из нее ветвью, начинающейся с помеченного единицей ребра. Множество N тех точек $\mathbb{R}^n$, в которых обращается в нуль числитель хотя бы одной из рациональных функций $F_v(x_1, x_2, \ldots, x_n)$, в силу свойства (R1) является замкнутым, причем это множество не может покрывать целиком ни один из секторов. Иначе в силу свойства (R2) произведение всех числителей рассматриваемых рациональных функций было бы нулевым полиномом, а это означало бы, что один из сомножителей этого произведения — нулевой полином.

Таким образом, каждое из множеств

$$S_i' = S_i \setminus N, \quad i = 1, 2, \ldots, n!,$$

есть непустое открытое множество. Покажем, что для массивов, соответствующих точкам из S_i', $i = 1, 2, \ldots, n!$, рассматриваемая сортировка в худшем случае требует не менее $\lceil \log_2 n! \rceil$ сравнений.

Будем говорить, что точка $(x_1, x_2, \ldots, x_n)$ некоторого сектора *согласуется* с листом l дерева D, если соответствующая дереву D сортировка массива $x_1, x_2, \ldots, x_n$ заканчивается в листе l.

В том случае, когда D имеет не менее $n!$ листьев, мы, рассуждая так же, как в примере 14.2, получаем, что высота дерева D (сложность сортировки по числу сравнений) не может быть меньше чем $\lceil \log_2 n! \rceil$. Допустим, что число листьев дерева D меньше чем $n!$. Тогда найдется лист l такой, что имеются точки в двух разных множествах S'_i, S'_j, согласующиеся с l. В силу свойства (R1) найдутся непустые открытые множества $U_1 \subset S'_i$, $U_2 \subset S'_j$ такие, что любая точка, принадлежащая какому-то одному из них, согласуется с листом l. Если листу l приписан массив (29.2), то для некоторых целых k, s, t, не меньших единицы и не превосходящих n, таких, что $s \neq t$, будем иметь $G_k(x_1, x_2, \ldots, x_n) = x_s$ при $(x_1, x_2, \ldots, x_n) \in U_1$ и $G_k(x_1, x_2, \ldots, x_n) = x_t$ при $(x_1, x_2, \ldots, x_n) \in U_2$. Но значения рациональной функции на одном из множеств U_1, U_2 однозначно определяют значения этой рациональной функции всюду на $U_1 \cup U_2$ (это выводится из свойства (R2)), поэтому из того, что, например, $G_k(x_1, x_2, \ldots, x_n) = x_s$ на U_1, следует, что $G_k(x_1, x_2, \ldots, x_n) = x_s$ на U_2. Но мы знаем, что $G_k(x_1, x_2, \ldots, x_n) = x_t$ на U_2. Так как $s \neq t$, то в любом секторе $x_s \neq x_t$, в частности, это так и в секторе, подмножеством которого является U_2. Противоречие. □

Из доказанного предложения следует, что $f(n) = n \log n$ является асимптотической нижней границей сложности алгоритмов сортировки массивов длины n попарно различных вещественных чисел с помощью сравнений и четырех арифметических операций. По теореме 29.1 эта функция является также асимптотической нижней оценкой для алгоритмов построения выпуклой оболочки.

Любой алгоритм построения выпуклой оболочки с помощью арифметических операций и сравнений, который имеет сложность $O(n \log n)$, является оптимальным по порядку сложности по общему числу операций, и его сложность есть $\Theta(n \log n)$. Алгоритм Грэхема построения выпуклой оболочки (*пример* 3.1) *является оптимальным по порядку сложности по общему числу операций, коль скоро используемая им сортировка имеет сложность $O(n \log n)$ по числу сравнений.*

§ 30. Классы P и NP

Цель этого и следующих параграфов — дать общее, без многих деталей, представление о некоторых проблемах, касающихся алгоритмов

полиномиально ограниченной сложности (см. § 5, определение 5.2) или, другими словами, полиномиальных алгоритмов.

Подход, согласно которому алгоритмы подразделяются на полиномиальные и все остальные, далеко не всегда является продуктивным с практической точки зрения (алгоритм со сложностью n^{100}, где n — размер входа, вряд ли найдет массовое применение), но при более широком взгляде этот подход имеет свою логику. Если для решения важной задачи никак не удается найти алгоритм, сложность которого хотя бы при каком-нибудь показателе d допускала бы оценку $O(n^d)$, то сам вопрос о существовании такого алгоритма нельзя назвать праздным.

Иногда бывает очень трудно установить принадлежность или соответственно непринадлежность задачи классу **P**, состоящему из таких задач, для которых существует полиномиальный алгоритм решения. В классе **P** выделяют подкласс P тех вычислительных задач, где результатом может быть лишь 1 или 0, т. е. фактически «да» или «нет», что типично для задач распознавания. К этим задачам относится, например, задача распознавания выполнимости булевой формулы, задача распознавания существования в графе гамильтонова цикла и т. д. Неизвестно, существуют ли полиномиальные алгоритмы их решения, но сравнительно легко устанавливается их принадлежность специальному классу NP. Определение класса NP будет дано ниже, из него будет следовать, что $\mathrm{P} \subset \mathrm{NP}$. Если бы было доказано, что $\mathrm{P} = \mathrm{NP}$, то в широком спектре случаев мы бы легко устанавливали принадлежность задачи распознавания классу P, т. е. устанавливали бы существование полиномиального алгоритма распознавания; его разработка — это отдельный вопрос. Но равенство $\mathrm{P} = \mathrm{NP}$ по сей день не доказано, и есть основания подозревать, что оно неверно. Возникает проблема $\mathrm{P} \stackrel{?}{=} \mathrm{NP}$, о которой мы тоже будем говорить.

Вначале скажем о ряде ограничений, налагаемых на рассматриваемые задачи. Во-первых, речь будет идти о задачах распознавания тех слов в данном алфавите Λ, которые принадлежат оговоренному подмножеству (языку) L множества всех слов Λ^* в этом алфавите. Все объекты, о которых идет речь в рассматриваемых задачах, должны быть представлены (закодированы) словами в алфавите Λ, входом алгоритма является одно-единственное слово в алфавите Λ. Размер входа x — это всегда длина $|x|$ (число букв) в слове x. Вычислительные затраты должны учитываться достаточно тщательно, неучтенной может оставаться лишь незначительная часть операций (во всяком случае, допускающая полиномиальную верхнюю оценку). Если имеется в виду модель вычислений PAM, то предполагается, что вычисле-

ния выполняются на базе конечного множества операций с тем или иным числом операндов, преобразующих буквы алфавита Λ в буквы алфавита Λ; исследуется сложность по числу этих операций — подобие битовой сложности. Требуется также учет числа присваиваний в ходе выполнения алгоритма.

Таким образом, содержательная задача распознавания (например, задача существования пути с фиксированными свойствами в заданном графе) должна быть для обсуждения ее принадлежности классам P и NP переформулирована в виде задачи распознавания принадлежности заданного слова некоторому языку. Такая формализация необходима потому что одна и та же математическая идея при одном из двух разных способов представления данных может приводить к полиномиальному алгоритму, а при другом — нет.

Часто при обсуждении проблемы $\mathrm{P} \stackrel{?}{=} \mathrm{NP}$ исходят из машины Тьюринга (МТ) как модели вычислений, при этом вычислительные затраты измеряются числом тактов работы машины. Это связано с тем, что модель МТ удобнее модели РАМ при доказательстве разного рода теорем об алгоритмах, в особенности — теорем о невозможности алгоритмов с теми или иными свойствами. Но модель РАМ ближе, чем МТ, к реальным вычислительным устройствам, и хотелось бы, чтобы результаты исследования классов P и NP, проводимые на основе модели МТ, сохраняли свою значимость для модели РАМ. Их значимость действительно сохраняется, о чем будет сказано в конце этого параграфа, а до той поры мы, как правило, не будем делать уточнений относительно модели вычислений. Но, повторяем, операции преобразования букв, входящих в слова, должны подсчитываться тщательно.

По сути дела, в каждой задаче распознавания принадлежности слова языку L обсуждается некоторый предикат $u(x)$, областью определения которого является все множество Λ^*, и $u(x) = 1$, если и только если $x \in L$. В дальнейшем нам часто будет удобно говорить не о языках, а о предикатах (по существу, это, конечно, одно и то же, но предикаты несколько предпочтительнее из-за того, что к ним можно применять логические операции, кванторы и т. д.). Соответственно, мы будем говорить о принадлежности предиката классу P и т. д. Иногда нам будет удобно рассуждать о предикатах не одной, а нескольких таких переменных, которые принимают значения в Λ^*. В этих случаях без специального упоминания предполагается, что в алфавит Λ включены некоторые разделители: если, скажем, разделитель «запятая» (т. е. «,») включен в Λ, то $v(x, y)$ имеет один аргумент x, y, где x и y суть слова в Λ^*, не содержащие разделителя «,» (кавычки по-

ставлены, чтобы не нарушать пунктуацию фразы). Соответственно, длиной пары слов x, y является $|x|+|y|+1$.

Теперь обсудим класс предикатов NP. Само его странное обозначение NP есть не что иное, как аббревиатура от английских слов nondeterministic polynomial — недетерминированный полиномиальный. Изначальное определение класса NP, появившееся в теории сложности в начале 70-х годов XX столетия, опиралось на понятие недетерминированного алгоритма[1]. Позднее появилось эквивалентное определение, не прибегающее к недетерминированности.

Определение 30.1. Заданный на Λ^* предикат $u(x)$ *принадлежит классу* NP, если существуют предикат $R(x,y)\in \mathrm{P}$ и полином $p(n)$ такие, что $u(x)$ можно представить в виде

$$u(x)=\exists_{y\in\Lambda^*,|y|\leqslant p(|x|)}R(x,y). \tag{30.1}$$

В том случае, когда $u(x)=1$, слово y называется *сертификатом* слова x.

Пример 30.1. Булева формула $f(x_1,x_2,\ldots,x_n)$ *выполнима*, если существуют значения $x_1^0,x_2^0,\ldots,x_n^0\in\{0,1\}$ переменных $x_1,x_2,\ldots,x_n$ такие, что

$$f(x_1^0,x_2^0,\ldots,x_n^0)=1;$$

в противном случае булева формула *невыполнима*. Так, булева формула

$$\neg(\neg x_1\vee x_2) \tag{30.2}$$

выполнима, так как ее значение при $x_1=1, x_2=0$ равно 1, а булева формула $x_1\wedge\neg x_1$ одной переменной x_1 невыполнима, так как и при $x_1=1$, и при $x_1=0$ значение этой формулы есть 0.

Любая булева формула может быть закодирована словом в алфавите

$$\Lambda_1=\{x,0,1,(,),\neg,\wedge,\vee\},$$

для этого переменные $x_1,x_2,x_3,\ldots$ кодируются как $x1,x10,x11,\ldots$: вслед за буквой x помещается номер переменной в двоичной системе. Формула (30.2) кодируется как $\neg(\neg x1\vee x10)$, при этом, например, слово $)x1x\neg$ в алфавите Λ_1 не является кодом какой-либо булевой формулы. Можно предложить простой алгоритм, который проверяет, является ли данное слово в алфавите Λ_1 кодом какой-либо булевой формулы, а также алгоритм, который по данному коду некой булевой формулы и данным булевым значениям (нулям и единицам) тех

[1] См., например, [13].

переменных, которые фактически присутствуют в этой формуле, находит ее значение. На основе этих двух алгоритмов можно построить алгоритм, который по данному слову $x \in \Lambda_1^*$ и слову y, являющемуся конечной последовательностью нулей и единиц, проверяет, верно ли, что

(a) x является кодом некоторой булевой формулы,

(b) значение закодированной словом x булевой формулы при тех значениях фактически присутствующих в ней переменных, которые последовательно указаны в слове y, равно 1.

(Число нулей и единиц в слове y равно числу переменных, фактически присутствующих в закодированной булевой формуле: например, в булевой формуле $x_3 \vee x_{15}$ фактически присутствуют две переменные, хотя для них и использованы большие индексы; сертификатом выполнимости этой формулы служит, например, слово 11.) Такой алгоритм можно сделать полиномиальным. Это говорит о том, что при указанном способе кодирования булевых формул предикат, распознающий выполнимость булевой формулы, принадлежит классу NP, — сертификатом является слово y, указывающее значения переменных, при которых значение булевой формулы равно 1, а значение предиката $R(x, y)$, входящего в (30.1), вычисляется с помощью алгоритма, проверяющего (a) и (b). Полином $p(n)$ можно взять равным n, так как $|y| \leqslant |x|$ в силу того, что y содержит значения только тех переменных, которые фактически присутствуют в булевой формуле.

Неизвестно, принадлежит ли классу P предикат на словах из Λ_1^*, распознающий выполнимость. Алгоритмы проверки выполнимости, основанные на переборе всех возможных комбинаций значений переменных $x_1, x_2, \ldots, x_n$, не являются полиномиальными.

Рассмотренный пример дает мотивацию определения 30.1 и проясняет смысл понятия «сертификат». Отметим два важных момента.

1) Перебор всех возможных слов y, $|y| \leqslant p(|x|)$, для проверки равенства $u(x) = 1$ с помощью (30.1) может потребовать рассмотрения экспоненциального по отношению к $|x|$ числа возможностей. Но определение 30.1 требует лишь, чтобы при $u(x) = 1$ сертификат существовал, вопрос же о сложности его построения не затрагивается.

2) Если $u \in$ NP, то равенство $u(x) = 0$ означает, что соответствующего сертификата для слова y не существует. Другими словами, принадлежность x множеству истинности предиката подтверждается сертификатом, а непринадлежность — отсутствием сертификата. По-

этому возможные значения 1 и 0 не являются симметричными или равноправными; как показано в примере 30.1, предикат, распознающий выполнимость булевой формулы, принадлежит NP, но мы не можем на основе только этого утверждать, что и предикат, распознающий невыполнимость булевой формулы, принадлежит NP (какое слово могло бы быть сертификатом невыполнимой формулы?).

Предложение 30.1. $\mathrm{P} \subseteq \mathrm{NP}$.

Доказательство. Определим $R(x, y) = u(x)$ в (30.1) при любом слове y и будем считать пустое слово сертификатом любого x такого, что $u(x) = 1$; в качестве полинома p берем нулевой полином. □

Продолжим рассмотрение примеров.

Пример 30.2. Согласно результатам Агравала, Кайала и Саксены (пример 22.2), классу P принадлежит предикат, определенный на словах в алфавите $\{0, 1\}$ и принимающий значение 1, если и только если слово является записью простого числа. Рассмотрим близкую задачу. Для заданных $n, k \in \mathbb{N}^+$, $k < n$, выяснить, имеется ли у числа n делитель l такой, что $1 < l \leqslant k$. Числа n и k задаются в двоичной системе; используя разделитель «;», мы можем кодировать пару n, k словом в алфавите

$$\Lambda_2 = \{0, 1, ;\}.$$

Если слово x кодирует указанным образом пару n, k, то в качестве сертификата можно взять двоичную запись некоторого конкретного делителя l, $1 < l < k$. Тогда входящий в (30.1) предикат $R(x, y)$ должен принимать значение 1, если и только если

(a) x является кодом пары n, k, удовлетворяющей сформулированным выше условиям,

(b) y является двоичной записью некоторого числа l такого, что $l \leqslant k$ и при этом $l \mid k$.

Существует алгоритм проверки условий (a), (b), вычислительные затраты которого ограничены величиной $C(|x| + |y|)^2$, где C — некоторая константа. Можно в (30.1) положить $p(n) = n$, так как $l \leqslant k$. Это говорит о том, что рассматриваемый предикат на Λ_2^* принадлежит классу NP. Неизвестно, принадлежит ли он также классу P.

Пример 30.3. В связи с изучением классов P и NP рассматривается большое число задач на графах, при этом графы, как правило, задаются матрицами смежности. В алфавите Λ_2 матрица смежности

кодируется словом, в котором строки матрицы выписаны одна за другой через разделитель «;». Легко видеть, что классу NP принадлежит предикат, принимающий значение 1, если и только если слово является кодом графа, обладающего *гамильтоновым циклом*, т. е. циклом, проходящим по одному разу через все вершины графа (рис. 24);

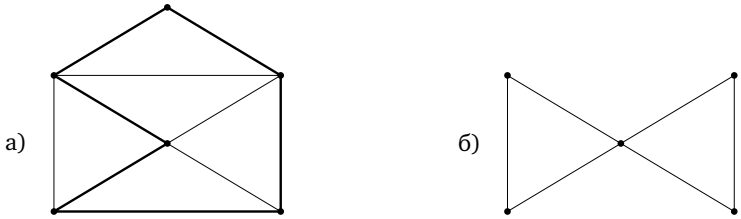


Рис. 24. а) Гамильтонов цикл; б) граф без гамильтонова цикла.

сертификатом может, например, являться последовательность двоичных номеров вершин гамильтонова цикла. Эти номера выписываются в одно слово через разделитель «;».

Аналогичным образом, классу NP принадлежит предикат, принимающий значение 1, если и только если слово является кодом графа, обладающего *кликой* с заданным числом m вершин. При этом кликой называется набор m вершин, любые две из которых соединены ребром (в изображенном на рис. 24а графе имеется несколько клик с тремя вершинами, но нет ни одной с четырьмя вершинами). Исходные данные кодируются словом в алфавите Λ_2: сначала идет матрица смежности, записанная так, как было сказано выше, затем, после разделителя «;», — число m в двоичной системе. Эта задача принадлежит классу NP, сертификатом может быть слово, составленное из двоичных номеров вершин клики, перечисленных через разделитель «;».

Относительно обоих этих предикатов на Λ_2^* неизвестно, принадлежат ли они классу P.

§ 31. Существование задач распознавания, не принадлежащих P. Связь моделей МТ и РАМ

Возникает вопрос, существуют ли вообще задачи распознавания принадлежности слов некоторому языку, разрешимые алгоритмически, но не принадлежащие P. Один из первых примеров задачи такого рода в 1973 г. был обнаружен М. Фишером и М. Рабином. Этот пример связан с так называемой арифметикой Пресбургера. Рассматривают-

ся логические формулы, записанные с соблюдением обычных синтаксических правил с помощью некоторого числа переменных, знаков $+, =$, логических связок $\vee, \wedge, \neg$ и скобок; при этом каждая переменная должна быть связана одним из кванторов $\forall, \exists$, а множеством значений переменных является $\mathbb{N}$. Каждая такая формула, трактуемая как утверждение о неотрицательных целых числах, имеет значение «истина» или «ложь» (1 или 0). Арифметика Пресбургера — это совокупность всех истинных формул указанного вида. Так, формулы

$$\forall_x \exists_y (x + y = x), \quad \forall_x \forall_y (x + y = y + x) \wedge \neg(\forall_x \exists_y (x + y = y))$$

принадлежат арифметике Пресбургера, а формула $\forall_x \exists_y (x + y = y)$ — нет. Как и ранее, мы будем использовать переменные, имеющие вид буквы x, за которой следует некоторое двоичное число (номер переменной). В алфавите

$$\Lambda_3 = \{x, 0, 1, +, =, \vee, \wedge, \neg, \forall, \exists, (,)\}$$

формула $\forall_x \exists_y (x + y = x)$ запишется в виде $\forall x1 \exists x10 (x1 + x10 = x1)$ и т. д. М. Пресбургером было в 1930 г. доказано существование алгоритма, который дает ответ на вопрос, является ли данное слово в алфавите Λ_3 формулой арифметики Пресбургера (разумеется, основная задача, решаемая этим алгоритмом, состоит в различении синтаксически правильных формул, принимающих значение 1 и соответственно 0; слова, не являющиеся синтаксически правильными формулами, легко распознаются и отвергаются).

Теорема, доказанная в 1973 г. Фишером и Рабином, может быть сформулирована так (мы не приводим доказательства [1]).

Теорема Фишера—Рабина. *Существует константа $c > 0$ такая, что для любого алгоритма A, распознающего среди всех слов из Λ_3^* те, которые являются формулами арифметики Пресбургера, существует целое n_0 такое, что для любого $n > n_0$ найдется слово из Λ_3^* длины n, для работы с которым алгоритму A потребуется более $2^{2^{cn}}$ вычислительных шагов.*

Из теоремы Фишера—Рабина следует, что арифметика Пресбургера как предикат на Λ_3^* не принадлежит классу P.

В доказательстве теоремы Фишера—Рабина вычислительные шаги соответствуют рассмотрению МТ в качестве модели вычислений.

С помощью модели МТ доказывается, например, и теорема о том, что если $t(n)$ — вычислимая с помощью некоторой машины Тьюринга

[1] См. [48].

функция натурального аргумента, принимающая натуральные значения, то существует такой язык в некотором алфавите, что сложность любого алгоритма (в виде машины Тьюринга) распознавания принадлежности слова этому языку будет больше $t(n)$ для всех достаточно больших n.[1] В определенном смысле эта теорема является более общей, чем теорема Фишера—Рабина, но она очень абстрактна. Теорема Фишера—Рабина примечательна тем, что в ней идет речь о некотором содержательном в математическом смысле языке (предикате).

Модель МТ, однако, выглядит избыточно затратной в сравнении с моделью РАМ в силу, например, того, что если некоторый символ хранится в ячейке a ленты и для определения хода дальнейших вычислений надо сравнить этот символ с символом, расположенным в ячейке b, которая находится далеко от ячейки a, то передвижение головки машины Тьюринга из a в b потребует большого числа тактов работы и, следовательно, больших затрат, а в случае модели РАМ сверх самой операции сравнения здесь нет затрат вообще.

Приведем соображение, которое можно оформить в виде теоремы[2], показывающее, что если некоторые вычисления имеют полиномиальную сложность при использовании РАМ, то они будут иметь полиномиальную сложность и при использовании МТ. Для простоты будем считать, что речь идет о преобразовании слов в алфавите $\Lambda = \{0, 1\}$ и соответственно о битовой сложности вычислений. Будем также считать, что на каждое присваивание тратится некоторое учитываемое время, в том числе на присваивание вида $a := b$, при котором не выполняется никаких сравнений и изменений бит. Это дает возможность предполагать, что для сложности $T(n)$ по числу битовых операций произвольного алгоритма и его пространственной сложности $S(n)$, измеряемой числом используемых дополнительных ячеек, в худшем случае выполняется соотношение $S(n) \leqslant C \cdot T(n)$ при некоторой положительной константе C.

Пусть $f : \Lambda^* \to \Lambda$ и $f_{\text{РАМ}}$ — некоторый алгоритм вычисления f с помощью РАМ, имеющий битовую сложность $T_{f_{\text{РАМ}}}(n)$, которая полиномиально ограничена по длине $n = |x|$ входа x (считаем, что в каждой ячейке РАМ размещается 0 или 1). Пусть $T_{f_{\text{РАМ}}}(n) < p(n)$, где p — некоторый полином. Тогда количество ячеек, используемых $f_{\text{РАМ}}$ при работе со словом x, сверх тех, в которых изначально хранится x, не превосходит $Cp(|x|)$ при некоторой положительной констан-

[1] См., например, [37, разд. 2], [4, разд. 4.2]. Впервые эта теорема была доказана, вероятно, М. Блюмом.

[2] См. [5, разд. 1.7].

те C. Если использовать МТ вместо РАМ, то возникающие дополнительные затраты на переходы от одной ячейки ленты к другой можно сделать меньшими, чем $|x| + Cp(|x|)$ при каждой битовой операции. Таким образом, общее число тактов работы МТ не превзойдет $p(|x|)(|x| + Cp(|x|))$, и сложность вычислений на МТ будет ограничена полиномом $p(n)(n + Cp(n))$.

§ 32. Полиномиальная сводимость. NP-полные задачи

Одна и та же математическая задача распознавания может быть формализована с использованием разных алфавитов и разных способов кодирования. Тем не менее, если речь, например, идет о некоторой арифметической задаче, то можно ожидать, что основание $q \in \mathbb{N}$, $q \geqslant 2$, выбранной позиционной системы счисления не будет влиять на принадлежность соответствующего предиката классу P, так как размеры $\lceil \log_{q_1}(n+1) \rceil$, $\lceil \log_{q_2}(n+1) \rceil$ записей числа n в двух разных системах счисления являются величинами одного порядка, и сам переход от одной записи к другой может быть выполнен за время, ограниченное полиномом от размера входа.

Такого рода связь может существовать и между совершенно разными задачами распознавания, формализованными с помощью предикатов на словах.

Определение 32.1. Пусть ν и $\tilde{\nu}$ — предикаты на словах в алфавитах Λ, $\widetilde{\Lambda}$. Говорят, что ν *полиномиально сводится* к $\tilde{\nu}$, и пишут $\nu \leqslant_p \tilde{\nu}$, если существует такая функция $f \in \mathbf{P}$, $f\colon \Lambda^* \longrightarrow \widetilde{\Lambda}^*$, что $\nu(x) = \tilde{\nu}(f(x))$.

Из того, что для $f \in \mathbf{P}$ длина слова $f(x)$ как функция от длины слова x полиномиально ограничена, а также из того, что композиция двух полиномов $p_3(t) = p_1(p_2(t))$ вновь является полиномом (при этом если $p_1(t), p_2(t)$ имеют неотрицательные коэффициенты, то и $p_3(t)$ тоже), мы получаем, что отношение $\leqslant_p$ транзитивно и что

$$\tilde{\nu} \in \mathrm{P} \implies \nu \in \mathrm{P}$$

при $\nu \leqslant_p \tilde{\nu}$.

В 70-х годах прошлого столетия С. Кук доказал замечательную теорему о существовании в NP таких предикатов на словах, к каждому из которых полиномиально сводится любой предикат из NP. Принятая терминология такова:

Определение 32.2. Предикат $u \in \mathrm{NP}$ называется NP-*полным*, если $\nu \leqslant_p u$ для каждого $\nu \in \mathrm{NP}$.

Перед формулировкой теоремы Кука напомним, что любую булеву формулу от $x_1, x_2, \ldots, x_n$ можно задать в конъюнктивной нормальной форме (КНФ):

$$D_1 \wedge D_2 \wedge \ldots \wedge D_m, \quad D_i = (l_{i1} \vee l_{i2} \vee \ldots \vee l_{i,k_i}), \quad i = 1, 2, \ldots, m, \qquad (32.1)$$

при этом l_{ij} является *литералом*, т. е. некоторой переменной x_s или переменной с отрицанием $\neg x_s$. Каждую формулу, заданную в КНФ, можно изобразить словом из Λ_1^*, как обсуждалось в примере 30.1. Предикат, принимающий значение 1 на тех и только тех словах из Λ_1^*, которые являются кодами КНФ выполнимых формул, назовем Sat (от английского слова satisfiability, одно из значений которого — выполнимость).

Теорему Кука мы приводим без доказательства[1]:

Теорема Кука. *Предикат* Sat *является* NP*-полным.*

Если показано, что некоторый предикат u является NP-полным, и при этом для некоторого другого предиката $v \in \mathrm{NP}$ установлено, что $u \leqslant_p v$, то из этого, очевидно, следует, что v тоже NP-полный.

Пример 32.1. Прямым следствием теоремы Кука является то, что рассмотренный в примере 30.1 предикат выполнимости произвольной, не обязательно заданной в КНФ, формулы является NP-полным. Полиномиальная сводимость предиката Sat к этому предикату очевидна: в качестве функции f, фигурирующей в определении 32.1, можно взять функцию, которая преобразует слово x из Λ_1^* в скобку «)» или еще в какой-нибудь символ, не являющийся кодом какой-либо булевой формулы, всякий раз, когда слово x не является кодом какой-либо КНФ, и преобразует x в x в противном случае (можно описать принадлежащий **P** алгоритм вычисления $f(x)$).

Когда говорят о принадлежащих классам P и NP задачах распознавания и о NP-полных задачах, а не о предикатах и языках, то при этом предполагают, что способ кодирования входных данных ясен из

[1] Мы опускаем доказательство этой теоремы, называемой в некоторых источниках также теоремой Кука—Левина, из-за того, что оно требует привлечения специфической техники доказательств утверждений о машинах Тьюринга; такого рода доказательства выглядят естественно в определенном контексте, который отличается от контекста этого курса. Отчасти это послужило и причиной того, что в § 31 мы оставили без доказательства теорему Фишера—Рабина. Доказательство теоремы Кука имеется, например, в [13], [4], [23]. Прозрачное изложение идей доказательства принадлежности классу NP предиката, распознающего выполнимость произвольной булевой формулы, и полиномиальной сводимости такого предиката к Sat можно найти в [10].

контекста и определен, по крайней мере, с точностью до полиномиальной сводимости. Обсуждая в дальнейшем задачи из примеров предыдущего параграфа, мы будем иметь в виду рассмотренные там способы кодирования.

Пример 32.2. Покажем сводимость задачи выполнимости КНФ к задаче о клике с указанным числом вершин (пример 30.3). Пусть КНФ F имеет вид (32.1). Построим граф G_F с $k_1 + k_2 + \ldots + k_m$ вершинами, для которых используем обозначения

$$l_{ij}, \quad i = 1, 2, \ldots, m, \quad j = 1, 2, \ldots, k_i,$$

при этом вершины l_{ij} и l_{vw} соединяем ребром, если и только если выполнены два условия:

- $i \neq v$,
- конкретные литералы, скрывающиеся в (32.1) за обозначениями l_{ij} и l_{vw}, не являются отрицанием друг друга (скажем, если l_{ij} — это $\neg x_1$, а l_{vw} — это x_1, то эти две вершины ребром не соединяются).

Построение матрицы смежности графа G_F может быть выполнено за полиномиально ограниченное время.

Из определения клики и способа построения графа G_F следует, что клика из m вершин в этом графе существует, если и только если формула (32.1) выполнима. В самом деле, при наличии такой клики полагаем $x_s = 0$, когда литерал $\neg x_s$ соответствует одной из вершин клики, а в остальных случаях $x_s = 1$. В результате каждое D_i в (32.1) равно 1.

Наоборот, если заданная формула (32.1) выполнима, то при соответствующих значениях всех переменных $x_1, x_2, \ldots, x_n$ для каждого $i \leqslant m$ можно выбрать j такое, что l_{ij} — это литерал со значением 1. Сделав такой выбор, мы получаем набор вершин, образующий клику с m вершинами.

Если исходная КНФ имеет вид

$$(\neg x_1 \vee \neg x_2) \wedge (x_2) \wedge (x_1 \vee x_2),$$

то мы получаем граф G_F с пятью вершинами (рис. 25), в котором обнаруживается клика (l_{11}, l_{21}, l_{32}) с тремя вершинами. Это соответствует тому, что исходная формула принимает значение 1 при $x_1 = 0$, $x_2 = 1$.

Задача распознавания существования в графе клики с заданным числом вершин является NP*-полной.*

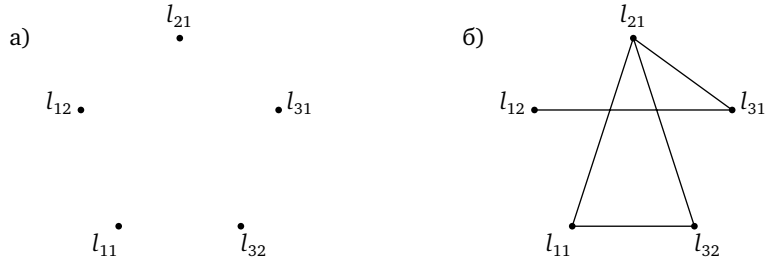


Рис. 25. Построение графа G_F для $F = (\neg x_1 \vee \neg x_2) \wedge (x_2) \wedge (x_1 \vee x_2)$: а) выбор вершин, б) проведение ребер.

Доказано также, что задача распознавания гамильтоновости графа является NP-полной[1]. Упомянем еще одну очень известную NP-полную задачу, называемую задачей о рюкзаке. Задано конечное множество U, размер $s(u) \in \mathbb{N}^+$ и стоимость $v(u) \in \mathbb{N}^+$ каждого $u \in U$, а также $a, b \in \mathbb{N}^+$. Существует ли такое подмножество $U' \subset U$, что $\sum\limits_{u \in U'} s(u) \leqslant a$, $\sum\limits_{u \in U'} v(u) \geqslant b$?

Из теоремы Кука следует, что для решения проблемы $\mathrm{P} \stackrel{?}{=} \mathrm{NP}$ достаточно со всей тщательностью рассмотреть какой-нибудь один NP-полный предикат, например, тот же предикат Sat, и ответить на вопрос о его принадлежности классу P. Если он принадлежит классу P, то $\mathrm{P} = \mathrm{NP}$ в силу NP-полноты рассматриваемого предиката, если не принадлежит, то $\mathrm{P} \neq \mathrm{NP}$, так как найден предикат, принадлежащий NP и не принадлежащий P. Но этот заманчивый план до сих пор реализовать не удалось, усилия многих исследователей не привели к решению этой проблемы, хотя и устоялось мнение, что, скорее всего, $\mathrm{P} \neq \mathrm{NP}$. Это предположение влечет за собой рекомендацию: если доказано, что решаемая практическая задача (при надлежащей формализации в виде предиката на словах в алфавите) является NP-полной, то было бы опрометчивостью рассчитывать на нахождение в короткие сроки полиномиального алгоритма ее решения, и лучше попробовать решить эту задачу приближенно.

Многие задачи фактического построения некоторого математического объекта вписываются в следующую схему: дано x; если существует y такое, что x вместе с y удовлетворяют фиксированному условию $R(x, y)$, то найти такое y. Соответствующая задача рас-

[1] Огромное число примеров NP-полных задач собрано в [13].

познавания выглядит так: дано x; требуется определить, существует ли y такое, что x вместе с y удовлетворяют фиксированному условию $R(x, y)$. Мы полагаем, что x и y — это коды некоторых математических объектов, т. е. слова в некотором алфавите Λ.

Пусть задача распознавания связана указанным образом с задачей построения, пусть $R(x, y) \in \mathrm{P}$ и полином p таков, что если существует какое-то решение задачи построения, то существует и такое решение y, что $|y| \leqslant p(|x|)$. Тогда задача распознавания принадлежит NP в соответствии с определением 30.1, причем в качестве сертификата для x выступает это решение y.

В примерах из § 30 по рассмотренным задачам распознавания легко восстанавливаются соответствующие им задачи построения (построить набор логических значений переменных; построить делитель данного числа; построить клику в графе, имеющую определенное количество вершин, и т. д.). Для доказательства принадлежности классу NP задачи распознавания мы брали в качестве сертификата само решение соответствующей вычислительной задачи.

Такой выбор сертификатов в ряде доказательств, видимо, и служит причиной довольно распространенного представления, что класс P образуют вычислительные задачи, решаемые за полиномиально ограниченное время, а класс NP — вычислительные задачи, для каждой из которых за полиномиально ограниченное время можно проверить, является ли данное слово y ее решением. На самом деле, конечно, в классы P и NP входят *только* задачи распознавания, но упомянутое представление, будучи, строго говоря, неправильным, в известной мере согласуется с реальным положением вещей.

Быстрый алгоритм распознавания наличия какого-то математического объекта, кодируемого словом из Λ^*, может в некоторых случаях позволить быстро решать и задачу фактического построения этого объекта. Проиллюстрируем это примером.

Пример 32.3. Мы знаем, что задача распознавания простоты натурального числа n принадлежит классу P, — алгоритм Агравала, Кайала и Саксены (пример 22.2) имеет битовую сложность $O(m^{11})$, где m — битовая длина n. Вопрос о существовании полиномиального алгоритма факторизации (разложения на простые множители) остается без ответа до сих пор при том, что алгоритмы факторизации имеют огромную важность, например, для криптографии. Вернемся в связи с этим вопросом к принадлежащей NP задаче, рассмотренной в примере 30.2: для заданных $n, k \in \mathbb{N}^+$, $k < n$, выяснить, имеется ли у числа n делитель l такой, что $1 < l \leqslant k$. Полиномиальный алгоритм реше-

ния этой задачи тоже неизвестен, но можно показать, что открытие такого алгоритма — назовем его A — автоматически дало бы полиномиальный алгоритм факторизации (кстати сказать, существование A автоматически следовало бы из равенства $\mathrm{P} = \mathrm{NP}$, если бы оно вдруг было доказано). Для этого можно воспользоваться бинарным поиском.

Пусть уже установлено, что n не имеет делителей, меньших n_1, где $n_1 < n$, и пусть n_2 таково, что $n_1 < n_2 < n$, и мы интересуемся наименьшим принадлежащим отрезку $[n_1, n_2]$ простым множителем числа n. Тогда, применяя A к n, n_3, где $n_3 = \left\lceil \frac{n_1 + n_2}{2} \right\rceil$, мы сузим диапазон поиска примерно вдвое: в зависимости от результата применения A мы перейдем от отрезка $[n_1, n_2]$ либо к отрезку $[n_1, n_3]$, либо к отрезку $[n_3 + 1, n_2]$. Первоначально же полагаем $n_1 = 2, n_2 = n$. Применив алгоритм A не более $m = \lceil \log_2(n+1) \rceil$ раз, мы найдем наименьший простой множитель t числа n. Повторяем те же вычисления для

$$n' = \frac{n}{t} \tag{32.2}$$

и т. д. Общее число простых множителей числа n с учетом их кратности ограничено сверху величиной $\log_2 n$, и, значит, величиной m. Битовые затраты каждого деления (32.2) не превосходят Cm^2, где C — некоторая константа. Если битовая сложность алгоритма A есть $O(m^d)$, то сложность описанного алгоритма факторизации будет допускать оценку $O(m^{d+2})$, т. е. этот алгоритм будет полиномиальным.

Задачи

145. Задача умножения квадратных булевых матриц линейно сводится к задаче построения транзитивно-рефлексивного замыкания (предполагается, что для сложностей рассматриваемых алгоритмов построения транзитивно-рефлексивного замыкания выполнено $T(3n) = O(T(n))$).

Указание. Пусть M_1 и M_2 — две булевы матрицы порядка n. Пусть X — булева матрица порядка $3n$:

$$\begin{pmatrix} 0 & M_1 & 0 \\ 0 & 0 & M_2 \\ 0 & 0 & 0 \end{pmatrix}.$$

Чему равны X^2, X^3? Воспользоваться формулой (23.1) для транзитивно-рефлексивного замыкания.

146. Здесь речь идет о линейной сводимости $P \leqslant Q$ задач, связанных с мультипликативными операциями над квадратными числовыми матрицами порядка n. Рассматриваются лишь такие алгоритмы решения задачи Q, для сложности по числу арифметических операций каждого из которых выполняется соотношение $T(kn) = O(T(n))$, $k = 2, 3$.

Требуется показать, что задача умножения произвольных квадратных матриц линейно сводится к задаче

а) умножения симметричных квадратных матриц;

б) умножения верхних треугольных матриц;

в) обращения невырожденных матриц.

Указание. Так же, как в предыдущей задаче, здесь можно прибегнуть к матрицам размера, большего n, используя исходные матрицы как блоки для построения новых матриц. В пункте в) полезно предварительно установить вид матрицы

$$\begin{pmatrix} I_n & M_1 & 0 \\ 0 & I_n & M_2 \\ 0 & 0 & I_n \end{pmatrix}^{-1},$$

где M_1, M_2 — исходные матрицы, I_n — единичная матрица порядка n.

147. а) Доказать свойство (R2) рациональных функций, сформулированное в § 29.

Указание. Достаточно доказать, что если полином $p(x_1, x_2, \ldots, x_n)$ тождественно равен нулю на некотором непустом открытом множестве $U \subset \mathbb{R}^n$, то этот полином нулевой. При $n = 1$ утверждение очевидно. Пусть $n > 1$ и точка $v = (v_1, v_2, \ldots, v_n) \in \mathbb{R}^n$ такова, что $p(v_1, v_2, \ldots, v_n) \neq 0$. Пусть $u \in U$. Множество U — открытое, поэтому у точки u существует окрестность некоторого радиуса $r > 0$, целиком принадлежащая U. Пусть l — расстояние от u до v, а $c_1, c_2, \ldots, c_n$ — координаты вектора единичной длины, направленного из u в v. Если t пробегает множество $\mathbb{R}$, то формулы

$$x_1 = u_1 + c_1 t, \quad x_2 = u_2 + c_2 t, \quad \ldots, \quad x_n = u_n + c_n t \tag{32.3}$$

задают прямую в $\mathbb{R}^n$, причем при $t = 0$ получается точка u, а при $t = l$ — точка v. Остается рассмотреть для полинома $\bar{p}(t)$ одной переменной t, получающегося подстановкой (32.3) в $p(x_1, x_2, \ldots, x_n)$, его значения в точке $t = l$ и на интервале $-r < t < r$.

б) Для каких целых $n \geqslant 1$ справедливо утверждение, что если произвольный полином с вещественными коэффициентами от $x_1, x_2, \ldots, x_n$ обращается в нуль на бесконечном подмножестве множества $\mathbb{R}^n$, то этот полином является нулевым?

148. Функция $f(n) = \lceil \log_2 n! \rceil$ является нижней границей сложности по числу сравнений алгоритмов сортировки массивов длины n

попарно различных рациональных чисел с помощью сравнений и четырех арифметических операций (в предложении 29.1 речь шла о сортировке вещественных чисел).

149. Функция $f(n) = \lceil \log_2(n+1) \rceil$ является нижней границей сложности по числу сравнений алгоритмов поиска места элемента в упорядоченном массиве длины n попарно различных вещественных чисел с помощью сравнений и четырех арифметических операций.

150. Пусть известен алгоритм, который по данным c, m, $c > 1$, $m \in \mathbb{N}^+$, строит m значащих двоичных цифр числа $\frac{1}{c}$ (построить m значащих цифр некоторого числа x, $0 < x < 1$, — это в данном контексте означает отыскать первую ненулевую цифру после запятой в двоичной записи этого числа, а затем отбросить все цифры после m цифр, отсчитанных от найденной), и пусть сложность этого алгоритма есть $O(f(m))$, где $f(m)$ — некоторая функция такая, что дополнительно известен алгоритм умножения произвольных $a, b \in \mathbb{N}^+$, сложность которого тоже есть $O(f(m))$, где $m = \max\{\lambda(a), \lambda(b)\}$. На основе этих двух алгоритмов сконструировать алгоритм построения частного и остатка от деления положительных целых a и b, имеющий сложность $O(f(m))$, $m = \max\{\lambda(a), \lambda(b)\}$.

Указание. Нужно построить q и r такие, что $a = qb + r$, $0 \leqslant r < b$, или $a\frac{1}{b} = q + s$, $0 \leqslant s < 1$. Возникновение погрешности при вычислении $\frac{1}{b}$ может привести к тому, что найденное q будет отличаться на 1 от точного значения; несколько добавочных проб помогут найти точные q и r, не изменяя оценки $O(f(m))$ для сложности.

151. Пусть $c > 0$ и y_0 удовлетворяет неравенствам $\frac{1}{2c} \leqslant y_0 \leqslant \frac{1}{c}$; пусть последовательность $y_0, y_1, y_2, \ldots$ получена по рекуррентной формуле

$$y_i = 2y_{i-1} - c_i y_{i-1}^2, \quad i = 1, 2, \ldots$$

Тогда последовательность $y_0, y_1, \ldots$ сходится к $\frac{1}{c}$.

152. (Продолжение предыдущей задачи.) Пусть $c \in \mathbb{N}^+$. Справедливо следующее утверждение[1]. Пусть y_0 удовлетворяет неравенствам $\frac{1}{2c} \leqslant y_0 \leqslant \frac{1}{c}$ и последовательность $y_0, y_1, y_2, \ldots$ получена по рекуррентной формуле

$$y_i = 2\tilde{y}_{i-1} - c_i \tilde{y}_{i-1}^2, \quad i = 1, 2, \ldots, \tag{32.4}$$

[1] См. [5, разд. 8.2].

где

- целое c_i таково, что $\lambda(c_i) = \lambda(c)$, и если $\lambda(c) > 2^i$, то первые 2^i цифр числа c_i совпадают с соответствующими цифрами числа c, а последующие цифры суть нули, если же $\lambda(c) \leqslant 2^i$, то $c_i = c$;
- $\tilde{y}_0$ получается из y_0 отбрасыванием всех цифр после первой значащей цифры;
- после вычисления значения y_i, $i > 0$, по рекуррентной формуле (32.4) в нем отбрасываются все цифры, идущие после первых 2^i значащих цифр, — это дает значение $\tilde{y}_i$.

Тогда первые $2^i - 3$ значащие цифры числа y_i совпадают с соответствующими цифрами числа $\frac{1}{c}$ при $i > 1$. Считая этот факт установленным и используя решение задачи 150, доказать, что задача деления одного целого числа на другое с остатком линейно сводится к задаче умножения двух целых чисел. (Размером входа считается $m = \max\{\lambda(a), \lambda(b)\}$, где a и b — исходные числа, при этом считаем, что m есть число вида 2^k; всюду подразумеваются битовые затраты; если это нужно, можно считать, что сложность $f(m)$ умножения удовлетворяет условиям $f(m) \leqslant f(2m) \leqslant 4f(m)$).

Указание. Соответствующий алгоритм построения частного и остатка уже описан в этой задаче и задаче 150. Достаточно доказать, что алгоритм приближенного обращения c, описанный в этой задаче, имеет сложность $R(2^k)$ такую, что $R(2^k) \leqslant \gamma f(2^k)$ для некоторой константы γ. Подобрать γ так, чтобы доказательство проводилось индукцией, и индуктивный переход был основан на неравенстве

$$R(2^k) \leqslant R(2^{k-1}) + 2f(2^{k-1}) + \delta 2^{k-1},$$

где константа δ определяется, в частности, тем, какой алгоритм сложения чисел используется.

153. Верно ли, что для доказательства того, что P = NP, достаточно показать, что хотя бы одна задача из NP принадлежит P?

154. Существуют ли в NP задачи, не являющиеся NP-полными?

155. Если бы оказалось, что полиномиального алгоритма распознавания простоты натурального числа не существует (забудем об алгоритме Агравала, Кайала и Саксены), то из этого бы следовало, что $P \neq NP$.

156. Дизъюнктивная нормальная форма (ДНФ) определяется как

$$C_1 \vee C_2 \vee \ldots \vee C_m, \quad C_i = (l_{i1} \wedge l_{i2} \wedge \ldots \wedge l_{ik_i}), \quad i = 1, 2, \ldots, m,$$

при этом каждое l_{ij} является литералом. Задача выполнимости ДНФ принадлежит P.

157. Найти ошибку или пробел в следующем доказательстве того, что $\mathrm{Sat} \in \mathrm{P}$. Очевидно, что любую КНФ можно преобразовать в эквивалентную ДНФ (см. задачу 156), поэтому задача выполнимости КНФ сводится к задаче выполнимости ДНФ, а эта задача принадлежит P.

158. Для выполнимой булевой формулы назовем соответствующий набор значений переменных *выполняющим*. Если $\mathrm{P} = \mathrm{NP}$, то существует полиномиальный алгоритм, который строит выполняющий набор для данной булевой формулы, если эта формула выполнима, и пустое слово, если формула невыполнима.

Приложение А

Основные алгоритмы сортировки и поиска

А1. Сортировка

Для простоты считаем, что требуется упорядочить по возрастанию числовой массив $x_1, x_2, \ldots, x_n$ с попарно различными элементами. Размер входа — число n. Мы называем *сегментом* массива $x_1, x_2, \ldots, x_n$ любую его часть $x_i, x_{i+1}, \ldots, x_{k-1}, x_k$, $1 \leqslant i \leqslant k \leqslant n$, которая по условию или по построению является упорядоченной.

1. Пузырьковая сортировка. Последовательным просмотром всех $x_1, x_2, \ldots, x_n$ определяется x_i такое, что $x_i > x_{i+1}$; затем x_i и x_{i+1} меняются местами, просмотр продолжается с элемента x_{i+1} и т. д. Тем самым в результате первого просмотра всего массива наибольший элемент передвинется на последнее место. Следующие просмотры начинаются опять сначала, после уменьшения на единицу количества просматриваемых элементов. Массив будет упорядочен после просмотра, который охватывал только первый и второй элементы, или же раньше, если при некотором просмотре не обнаружено x_i такого, что $x_i > x_{i+1}$.

2. Сортировка выбором. Выполняется $n - 1$ шаг. На i-м шаге ($i = 1, 2, \ldots, n - 1$) среди элементов $x_i, x_{i+1}, \ldots, x_n$ отыскивается наименьший и переставляется с x_i.

3. Сортировка простыми вставками (два варианта). Пусть после нескольких шагов сортировки элементы $x_1, x_2, \ldots, x_i$ уже упорядочены (образуют сегмент): $x_1 < x_2 < \ldots < x_i$. Тогда на следующем шаге элемент x_i вставляется в этот сегмент таким образом, что элементы $x_1, x_2, \ldots, x_{i+1}$ оказываются упорядоченными (сегмент расширяется). В конечном счете получаем сегмент $x_1, x_2, \ldots, x_n$. В первом варианте сортировки место вставки определяется последовательными сравнениями x_{i+1} с $x_i, x_{i-1}, \ldots$, во втором — последовательными сравнениями x_{i+1} с $x_1, x_2, \ldots$

4. Сортировка бинарными вставками. Отличается от сортировки простыми вставками тем, что место x_i в сегменте $x_1, x_2, \ldots, x_{i-1}$ определяется алгоритмом бинарного поиска (см. А2, п. 4).

5. Сортировка слияниями. Разнообразные виды этой сортировки используют слияние сегментов. Сначала мы рассмотрим процедуру слияния, а затем опишем два варианта сортировки, основанной на этой процедуре.

Слияние. Пусть для элементов массива $e_1, e_2, \ldots, e_m$ выполнено $e_1 < e_2 < \ldots < e_k$ и $e_{k+1} < e_{k+2} < \ldots < e_m$, $k \leqslant m$. Массив $f_1, f_2, \ldots, f_m$, который является результатом слияния массивов $e_1, e_2, \ldots, e_k$ и $e_{k+1}, e_{k+2}, \ldots, e_m$, можно получить за m шагов. После i-го шага элементы $f_1, f_2, \ldots, f_i$ уже имеют нужные значения, целые p и q $(p + q = i)$ показывают, сколько элементов из числа $e_1, e_2, \ldots, e_k$ и $e_{k+1}, e_{k+2}, \ldots, e_m$ уже использовано.

Рекурсивный вариант сортировки слияниями. При $n = 1$ массив упорядочен. Пусть $n > 1$, тогда массив $x_1, x_2, \ldots, x_n$ разбивается на два примерно равных по длине подмассива $x_1, x_2, \ldots, x_{\lfloor n/2 \rfloor}$ и $x_{\lfloor n/2 \rfloor + 1}, x_{\lfloor n/2 \rfloor + 2}, \ldots, x_n$. Сортировка применяется рекурсивно к этим подмассивам, после чего выполняется слияние.

Сортировка фон Неймана. Первоначально элементы массива $x_1, x_2, \ldots, x_n$ рассматриваются как упорядоченные одноэлементные сегменты. Затем в массиве $y_1, y_2, \ldots, y_n$ образуются упорядоченные сегменты длины 2, получающиеся слиянием x_1 и x_2, x_3 и x_4, x_5 и x_6, ... Последний сегмент будет иметь один или два элемента в зависимости от четности n. Полученные сегменты сливаются в упорядоченные сегменты длины 4 (кроме последнего, который тоже упорядочен, но, возможно, имеет длину 1, 2 или 3), они последовательно попадают в массив $x_1, x_2, \ldots, x_n$. Процесс укрупнения сегментов продолжается дальше. В некий момент массив $x_1, x_2, \ldots, x_n$ или $y_1, y_2, \ldots, y_n$ содержит только один упорядоченный сегмент.

6. Быстрая сортировка. Эта сортировка основывается на процедуре разбиения массива. Перед описанием сортировки мы рассмотрим эту процедуру.

Разбиение. Берется первый элемент массива и сравнивается со всеми остальными. Меньшие его элементы помещаются в начальную часть массива, бóльшие — в конечную. Сам первоначально взятый элемент помещается между этими двумя частями, это — то место, которое ему надлежит занимать в упорядоченном массиве. Дополнительный массив для этой процедуры не требуется, достаточно двух переменных p и q, показывающих, сколько элементов в начальной

и конечной частях уже занято. Элемент, взятый первым, расположен на $(p+1)$-м месте и сравнивается со следующим за ним. Равенство $p+q=n-1$ означает, что разбиение завершено.

Сортировка. Выполняется разбиение; в результате элемент, ранее располагавшийся в массиве первым, занимает нужное место (с некоторым номером k, $1 \leqslant k \leqslant n$). Затем быстрая сортировка применяется рекурсивно к сегментам $x_1, x_2, \ldots, x_{k-1}$ и $x_{k+1}, x_{k+2} \ldots, x_n$.

А2. Поиск

Числовой массив $x_1, x_2, \ldots, x_n$ имеет попарно различные элементы. В п. 4 элементы предполагаются упорядоченными по возрастанию.

1. Поиск наименьшего. Просматриваются последовательно x_2, $x_3, \ldots, x_n$ и каждый новый элемент x_i сравнивается с уже найденным наименьшим среди $x_1, x_2, \ldots, x_{i-1}$.

2. Поиск m-го наименьшего. Элементы $x_1, x_2, \ldots, x_n$ переставляются в соответствии с процедурой разбиения (см. алгоритм быстрой сортировки). Пусть элемент, бывший в исходном массиве первым, после выполнения процедуры стал k-м, $1 \leqslant k \leqslant n$. Если $m=k$, то задача решена. Если $m<k$, то разыскивается m-е наименьшее среди $x_1, x_2, \ldots, x_{k-1}$; если $m>k$, то разыскивается $(m-k)$-е наименьшее среди $x_{k+1}, x_{k+2}, \ldots, x_n$.

3. Одновременный поиск наименьшего и наибольшего. Элементы $x_1, x_2, \ldots, x_n$ просматриваются последовательными парами: x_1, x_2, затем x_3, x_4 и т. д. (последний элемент может остаться без пары). При рассмотрении k-й пары x_{2k-1}, x_{2k} в ней выбираются наименьший и наибольший элементы, которые сравниваются с уже найденными наименьшим и, соответственно, наибольшим среди $x_1, x_2, \ldots, x_{2k-2}$. Если n нечетно, то на последнем шаге x_n сравнивается с уже найденными наименьшим и наибольшим среди $x_1, x_2, \ldots$ $\ldots, x_{n-1}$.

4. Бинарный поиск места элемента. Кроме упорядоченного массива $x_1 < x_2 < \ldots < x_n$ дано число y, для которого априори может осуществляться любая из возможностей

$$y \leqslant x_1, \quad x_1 < y \leqslant x_2, \quad \ldots, \quad x_{n-1} < y \leqslant x_n, \quad x_n < y.$$

Этим возможностям присваиваются номера $1, 2, \ldots, n+1$. Требуется найти номер фактически осуществившейся возможности. Первоначальный диапазон поиска — от 1 до $n+1$. Каждый шаг би-

нарного поиска сужает диапазон примерно вдвое: если перед очередным шагом диапазон был от p до q, то y сравнивается с x_r, $r = \lfloor (p+q)/2 \rfloor$. При $x_r < y$ диапазон дальнейшего поиска — от $r+1$ до q (в дальнейшем рассматривается сегмент $x_{r+1}, x_{r+2}, \ldots, x_{q-1}$), в противном случае — от p до r (в дальнейшем рассматривается сегмент $x_p, x_{p+1}, \ldots, x_r$). И так далее до совпадения границ диапазона.

Приложение B

Оценивание сумм значений монотонных функций

Предложение B.1. *Пусть f — неубывающая или невозрастающая непрерывная на отрезке $[n_0, n_1]$ функция, $n_0, n_1 \in \mathbb{Z}$. Тогда*

$$\int_{n_0}^{n_1} f(x)\,dx + f(n_0) \leqslant \sum_{k=n_0}^{n_1} f(k) \leqslant \int_{n_0}^{n_1} f(x)\,dx + f(n_1) \qquad \text{(B.1)}$$

в случае неубывающей f и

$$\int_{n_0}^{n_1} f(x)\,dx + f(n_1) \leqslant \sum_{k=n_0}^{n_1} f(k) \leqslant \int_{n_0}^{n_1} f(x)\,dx + f(n_0) \qquad \text{(B.2)}$$

в случае невозрастающей f.

Доказательство. Пусть f — неубывающая функция. Рис. 26 показывает, что $\sum\limits_{k=n_0}^{n_1-1} f(k) \leqslant \int_{n_0}^{n_1} f(x)\,dx$ (значение $\sum\limits_{k=n_0}^{n_1-1} f(k)$ равно сумме

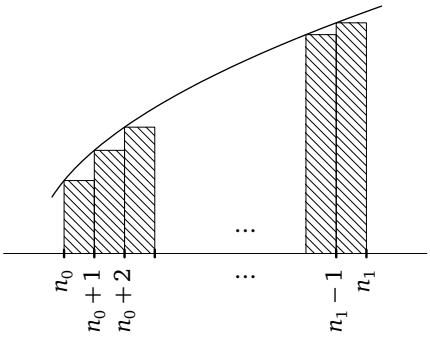


Рис. 26. $\sum\limits_{k=n_0}^{n_1-1} f(k) \leqslant \int_{n_0}^{n_1} f(x)\,dx$.

площадей выделенных прямоугольников с вертикальными сторонами $f(n_0), f(n_0+1), \ldots, f(n_1-1)$). Соответственно, рис. 27 показывает,

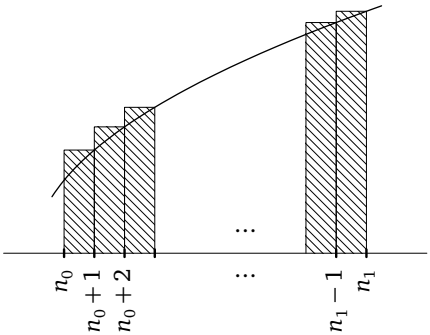


Рис. 27. $\sum\limits_{k=n_0+1}^{n_1} f(k) \geqslant \int_{n_0}^{n_1} f(x)\,dx$.

что $\sum\limits_{k=n_0+1}^{n_1} f(k) \geqslant \int_{n_0}^{n_1} f(x)\,dx$ (значение $\sum\limits_{k=n_0+1}^{n_1} f(k)$ равно сумме площадей выделенных прямоугольников с вертикальными сторонами $f(n_0+1), f(n_0+2), \ldots f(n_1)$). Это дает нам (B.1). Неравенства (B.2) доказываются аналогично. □

Рассмотрим некоторые примеры. Функция $\sqrt{x}$ не убывает на правой полуоси. Имеем

$$\int_1^n \sqrt{x}\,dx + 1 \leqslant \sum_{k=1}^{n} \sqrt{k} \leqslant \int_1^n \sqrt{x}\,dx + \sqrt{n},$$

т. е.

$$\frac{2}{3}(\sqrt{n})^3 + \frac{1}{3} \leqslant \sum_{k=1}^{n} \sqrt{k} \leqslant \frac{2}{3}(\sqrt{n})^3 + \sqrt{n} - \frac{2}{3},$$

и, как следствие,

$$\sum_{k=1}^{n} \sqrt{k} = \frac{2}{3}(\sqrt{n})^3 + O(\sqrt{n}).$$

Аналогично выводится, что при любом вещественном $\alpha \geqslant 0$ и $n \to \infty$ выполняется $\sum\limits_{k=0}^{n} k^\alpha \sim \dfrac{n^{\alpha+1}}{\alpha+1}$. (Справедлива ли эта формула при всех $\alpha \neq -1$?)

Функция $\frac{1}{x}$ не возрастает при $x \geqslant 1$. Имеем

$$\int_1^n \frac{dx}{x} + \frac{1}{n} \leqslant \sum_{k=1}^{n} \frac{1}{k} \leqslant \int_1^n \frac{dx}{x} + 1,$$

это дает нам

$$\ln n \leqslant \sum_{k=1}^{n} \frac{1}{k} \leqslant \ln n + 1,$$

и, как следствие,

$$\sum_{k=1}^{n} \frac{1}{k} = \ln n + O(1).$$

Приложение C

Проблема орбит

C1. Показательная функция и логарифмическая оценка

В этом разделе приложения рассказывается об одной вычислительной задаче и об эффективном решении ее некоторого частного случая; почти все факты приводятся без доказательств. В разделе C2 мы рассмотрим оставшийся случай с бóльшими подробностями.

Речь идет об одном из вариантов известной «проблемы орбит». Даны две квадратные матрицы A и B одинакового порядка с элементами из поля $\mathbb{Q}$. Существуют ли натуральные m такие, что $A^m = B$? Если да, то надо указать все такие m. Рассмотрение собственных значений матриц A и B приводит к вопросу о распознавании существования и поиске натуральных m таких, что

$$a^m = b \tag{C.1}$$

для данных алгебраических чисел a и b.

Напомним, что число $a \in \mathbb{C}$ называется *алгебраическим*, если существует полином над полем $\mathbb{Q}$, для которого a является корнем. В качестве такого полинома удобно рассматривать *унитарный* [1] полином, т. е. полином с единичным старшим коэффициентом. Можно показать, что для всякого алгебраического числа существует единственный унитарный неприводимый над $\mathbb{Q}$ полином, для которого a является корнем. Алгебраическое число $a \in \mathbb{C}$ называется *целым алгебраическим*, если соответствующий неприводимый унитарный полином является полиномом над $\mathbb{Z}$, т. е. имеет целые коэффициенты [2]. Нетрудно, например, убедиться, что число $(1+\sqrt{5})/2$ — целое алгебраическое, а число $(3+4\sqrt{-1})/5$ — нецелое алгебраическое.

[1] Используются также термины *приведенный*, *нормированный*.

[2] Для избежания терминологической путаницы между целыми алгебраическими числами и обычными целыми, т. е. элементами $\mathbb{Z}$, последние в теории алгебраических чисел часто называют *целыми рациональными* числами.

Если $|a| \neq 1$, то с исследованием и решением уравнения (С.1) серьезных трудностей не возникает, так как модуль значения показательной функции a^m монотонно возрастает или монотонно убывает, и можно получить логарифмическое ограничение на m. Задача становится интересной при $|a| = 1$ и при дополнительном условии, что a не является корнем из 1. (Если a есть корень из единицы, то задача тривиальна.)

Как это следует из элементарной теории алгебраических чисел, вопрос можно переформулировать так: дан неприводимый над $\mathbb{Q}$ полином $f(x)$, $\deg f(x) = n$, и некоторый полином $q(x)$ над $\mathbb{Q}$ степени $\deg q(x) < n$; существуют ли такие натуральные m, что

$$a^m = q(a) \tag{С.2}$$

при любых $a \in \mathbb{C}$, для которых $f(a) = 0$, и если да, то как найти такие m? Мы сосредоточимся на последнем вопросе, опуская мелкие подробности перехода к нему от проблемы орбит.

Особенность постановки вопроса об уравнении (С.2) состоит в том, что речь идет не об одном фиксированном корне полинома $f(x)$, но обо всех его корнях. Легко показать, что если равенство (С.2) верно для одного какого-то корня a_0 полинома $f(x)$, то оно верно для всех корней этого полинома. В самом деле, если для некоторого фиксированного m полиномы $x^m - q(x)$ и $f(x)$ имеют общий корень a_0, то эти полиномы имеют нетривиальный общий делитель. Наибольший общий делитель этих полиномов может быть найден алгоритмом Евклида и, следовательно, должен иметь рациональные коэффициенты. Из этого и из неприводимости $f(x)$ над $\mathbb{Q}$ следует, что наибольший общий делитель равен $f(x)$. Отсюда получаем, что $x^m - q(x)$ делится на $f(x)$. Это означает, что каждый корень $f(x)$ является также корнем $x^m - q(x)$.

Таким образом, задача о всей совокупности корней эквивалентна задаче об одном фиксированном корне, и задача об одном фиксированном корне эквивалентна задаче о любом другом корне. Это обстоятельство позволяет справиться со случаем целого алгебраического a. К решению подводит серия результатов о корнях унитарных полиномов над $\mathbb{Z}$, начавшаяся с работ Кронекера и завершившаяся в 60—70-е годы XX века исследованиями А. Шинцеля, Г. Цассенхауза, П. Бланксби и Х. Монтгомери[1]. Выяснено, что если унитарный полином над $\mathbb{Z}$ степени n таков, что его корни не являются корнями из

[1] См. [58], [45]. Результаты Кронекера, Шинцеля и Цассенхауза изложены в [29, разд. 20.2].

единицы, то у него найдется по меньшей мере один корень a, для которого $|a|>1$. Бланксби и Монтгомери показали, что по меньшей мере для одного корня выполняется неравенство

$$|a| \geqslant 1+\frac{1}{30n^2\ln(6n)}. \qquad \text{(C.3)}$$

Из этого позднее было выведено, что если корни неприводимого унитарного полинома $f(x)$ степени n с целочисленными коэффициентами не являются корнями из единицы, то для любого натурального решения m уравнения (C.2) выполнено неравенство

$$m \leqslant n+60n^2\ln(6n)(\ln(n+1)+\ln|q(x)|), \qquad \text{(C.4)}$$

где $|q(x)|$ обозначает длину вектора коэффициентов полинома $q(x)$: если $q(x)=c_kx^k+\ldots+c_1x+c_0$, то

$$|q(x)|=\sqrt{c_k^2+\ldots+c_1^2+c_0^2}.$$

Решения уравнения (C.2) для различных корней a полинома $f(x)$ совпадают. Доказанное может быть сформулировано следующим образом.

Предложение C.1. *Пусть $f(x)$ — неприводимый над $\mathbb{Q}$ полином степени n, корни которого не являются корнями из единицы. Пусть $q(x)$ — некоторый полином над $\mathbb{Q}$, степень которого меньше n. Тогда для любого корня a полинома $f(x)$ существует не более одного натурального решения уравнения* (C.2)*, при этом решения m уравнения* (C.2) *для различных корней a полинома $f(x)$ совпадают, и имеет место оценка* (C.4).

Несложно показать, что верны следующие утверждения:

1. Пусть $h(x)\in\mathbb{Q}$ и $r(x)$ — остаток от деления $h(x)$ на $f(x)$. Пусть $a\in\mathbb{C}$, $f(a)=0$. Тогда $h(a)=r(a)$.

2. Пусть $q_1(x), q_2(x), q_3(x), \ldots$ суть остатки от деления $x, x^2, x^3, \ldots$ на $f(x)$. Тогда

- при $k>1$ полином $q_k(x)$ равен остатку от деления $xq_{k-1}(x)$ на $f(x)$;
- равенство (C.2) имеет место, если и только если $q(x)=q_m(x)$.

Из этих утверждений следует, что после того, как установлена граница M для возможного значения m, распознавание существования и поиск натурального решения уравнения (C.2) можно выполнить, затратив $O(nM)$ арифметических операций над рациональными числами.

С2. Неудачно выбранный размер входа

Обратимся к уравнению (С.2), предполагая, что среди коэффициентов унитарного неприводимого над $\mathbb{Q}$ полинома $f(x)$ имеется по крайней мере один нецелый рациональный; без специальных оговорок считаем, что корни $f(x)$ не являются корнями из единицы. Возникает вопрос: можно ли и для этого случая получить подобную (С.4) верхнюю оценку величины m? На протяжении ряда лет считалось, что ответ положительный [1] и что существует полином трех переменных $P(x_1, x_2, x_3)$ такой, что даже при наличии среди коэффициентов $f(x)$ нецелых рациональных чисел имеет место оценка

$$m < P(n, \log_2|f|, \log_2|q|), \tag{С.5}$$

но затем выяснилось, что это неверно [2]. Проблема заслуживает детального рассмотрения, поскольку ряд ее моментов является принципиальным. Вывод о существовании полинома $P(x_1, x_2, x_3)$ был сделан на основании утверждения о том, что если среди коэффициентов $f(x)$ есть нецелые числа, то имеет место неравенство

$$m \leqslant (n-1)\log_2|f|. \tag{С.6}$$

Неравенство (С.6) опровергается несложно (сразу заметно, что правая часть этого неравенства не зависит от вида полинома $q(x)$, что подозрительно). Мы, тем не менее, сосредоточим усилия на другом — покажем, что верхняя оценка (С.5) не может иметь места не только при полиномиальной, но и, например, при показательной функции того или иного вида (скажем, вида $2^{x_1^{x_2^{x_3}}}$) и вообще при любой непрерывной функции $P(x_1, x_2, x_3)$. Это, в частности, лишает возможности характеризовать вычислительную сложность алгоритма как полиномиальную, экспоненциальную и т. д.

Указанное обстоятельство в значительной мере является следствием плохо выбранных параметров n, $|f(x)|$ и $|q(x)|$ размера входа. Если мы зафиксируем n и $f(x)$, а затем будем слегка изменять значения коэффициентов полинома $q(x)$ (тем самым два параметра размера остаются фиксированными, а третий изменяется очень мало), то при этом могут возникать уравнения вида (С.2), имеющие натуральные решения m, и эти m для разных уравнений могут, как будет показано, отличаться друг от друга сколь угодно сильно. Мы укажем натуральное n, неприводимый полином $f(x) \in \mathbb{Q}[x]$ степени n,

[1] См. [51]; в этой же работе получена оценка (С.4) как следствие оценки (С.3).
[2] См. [41].

последовательность $q_1(x), q_2(x), \ldots \in \mathbb{Q}[x]$ полиномов степени меньшей, чем n, и вещественные A, B, $A < B$, такие, что $A < |q_k(x)| < B$, $k = 1, 2, \ldots$, и при этом каждое из уравнений

$$a^m = q_k(a), \quad f(a) = 0, \tag{C.7}$$

имеет решение $m = k$. Возьмем

$$a = \frac{3 + 4\sqrt{-1}}{5}, \tag{C.8}$$

тогда $f(x) = x^2 - \frac{6}{5}x + 1$, $n = 2$ и $|f(x)| = \sqrt{1 + \frac{36}{25} + 1} = \frac{\sqrt{86}}{5}$. Пусть $b_1, b_2, \ldots$ и $c_1, c_2, \ldots$ — последовательности рациональных чисел таких, что $a^k = b_k a + c_k$, и пусть $q_k(x) = b_k x + c_k \in \mathbb{Q}[x]$ (таким образом, $q(x)$ равен остатку от деления x^k на $f(x) = x^2 - \frac{6}{5}x + 1$). Очевидно, что $b_1 = 1$, $c_1 = 0$.

Используя утверждения 1, 2, сформулированные в конце предыдущего раздела этого приложения, индукцией по k легко доказывается, что

$$b_{k+1} = \frac{6}{5}b_k + c_k, \quad c_{k+1} = -b_k$$

для $k > 0$. Последняя система двух рекуррентных уравнений решается легко, так как второе уравнение позволяет заменить в первом c_k на $-b_{k-1}$ и получить для b_k линейное рекуррентное уравнение второго порядка с постоянными коэффициентами. В итоге имеем (см. приложение G)

$$b_k = \frac{5}{8}\sqrt{-1}\left(\left(\frac{3 - 4\sqrt{-1}}{5}\right)^k - \left(\frac{3 + 4\sqrt{-1}}{5}\right)^k\right) = \frac{5}{4}\sin(k\theta)$$

и

$$c_k = -\frac{5}{8}\sqrt{-1}\left(\left(\frac{3 - 4\sqrt{-1}}{5}\right)^{k-1} - \left(\frac{3 + 4\sqrt{-1}}{5}\right)^{k-1}\right) = -\frac{5}{4}\sin((k-1)\theta),$$

где $\theta \in \mathbb{R}$ выбрано так, чтобы выполнялось $a = e^{\theta\sqrt{-1}}$. Получаем

$$|q_k| = \sqrt{b_k^2 + c_k^2} = \frac{5}{4}\sqrt{\sin^2 k\theta + \sin^2(k-1)\theta} \le \frac{5}{4}\sqrt{2}.$$

Для функции $h(x) = \sqrt{\sin^2 x + \sin^2(x - \theta)}$ положим $\mu = \min\limits_{0 \le x < 2\pi}(h(x)) \ge 0$. Так как $\sin(\theta) = \frac{4}{5}$, то θ не является целым кратным π, поэтому $\mu > 0$. Таким образом,

$$|q_k| = \frac{5}{4}h(k\theta) \ge \frac{5}{4}\mu > 0.$$

Положим $A = \frac{5}{4}\mu, B = \frac{5}{4}\sqrt{2}$. Для любой функции $F(x_1, x_2, x_3)$, непрерывной на множестве

$$V = \left\{(x_1, x_2, x_3) : x_1 = 2, x_2 = \frac{\sqrt{86}}{5}, A < x_3 < B\right\},$$

мы можем выбрать целое k, большее максимума функции $F(x_1, x_2, x_3)$ на V, и это приведет к уравнению (C.7) с решением m, большим $F(n, |f(x)|, |q(x)|)$.

Таким образом, нами доказана следующая теорема.

Теорема C.1. *Пусть алгоритм распознавания существования и поиска решения m уравнения* (C.2) *состоит в получении верхней границы M для m и в последующей проверке всех значений $m = 1, 2, \ldots, M$. Тогда сложность этого алгоритма по числу таких проверок не допускает оценки сверху вида $F(n, |f(x)|, |q(x)|)$, где F — какая-либо непрерывная функция трех переменных.*

Отсутствие непрерывной функции, ограничивающей затраты, является, как уже упоминалось, следствием неудачно выбранных параметров размера входа. Можно показать[1], что если среди коэффициентов унитарного неприводимого полинома $f(x)$ имеется хотя бы один, не являющийся целым рациональным, и если $C, D \in \mathbb{Z}$ таковы, что Ca является целым алгебраическим для любого корня a полинома $f(x)$ и $Dq(x) \in \mathbb{Z}[x]$, то для решения m уравнения (C.2) выполнено

$$m \leqslant (n-1)\log_2 C + \log_2 D. \qquad \text{(C.9)}$$

Границы (C.4) и (C.9) приводят к алгоритму решения проблемы орбит.

В качестве C может быть взято наименьшее общее кратное знаменателей коэффициентов полинома $f(x)$, в качестве D — наименьшее общее кратное знаменателей коэффициентов полинома $q(x)$.

Неверно, что C всегда можно взять меньшим[2], чем $|f(x)|$. В этом можно убедиться, вернувшись к $f(x) = x^2 - \frac{6}{5}x + 1$.

[1] См. уже упоминавшуюся работу [41].

[2] В [51] авторы исходили из предположения, что всегда можно взять $C < |f(x)|$.

Приложение D

Оптимальность схемы Горнера

Теорема D.1. *Пусть n — произвольное неотрицательное целое число. Не существует алгоритма, который, используя только сложение, вычитание и умножение, позволяет вычислять значение*

$$a_n x^n + \ldots + a_1 x + a_0 \tag{D.1}$$

по числам $x, a_0, a_1, \ldots, a_n$ так, что количество сложений или умножений всегда оказывается меньше n.

В терминах нижних границ это утверждение можно, очевидно, переформулировать следующим образом.

Пусть $\mathscr{P}$ — класс алгоритмов, вычисляющих значение (D.1) *с помощью операций сложения, вычитания и умножения. Будем рассматривать число n как размер входа $x, a_0, a_1, \ldots, a_n$. Тогда n является нижней границей как аддитивной сложности* (*измеряемой числом сложений и вычитаний в худшем случае*), *так и мультипликативной сложности* (*измеряемой числом умножений в худшем случае*) *алгоритмов класса $\mathscr{P}$.*

Несколько предварительных соглашений и замечаний. Во-первых, будем для определенности считать, что все коэффициенты полинома, а также значение x принадлежат полю рациональных чисел $\mathbb{Q}$, хотя достаточно было бы потребовать, например, чтобы они принадлежали произвольному бесконечному полю. Во-вторых, ограничимся операциями сложения и умножения; позднее мы покажем, что привлечение вычитаний не является существенным. В-третьих, заметим, что любой алгоритм, который использует только операции сложения и умножения и который вычисляет значение произвольного полинома фиксированной степени n по заданному x, можно записать в виде линейной (неветвящейся) программы, одной и той же для всех полиномов степени n (так как в используемом наборе операций нет операций сравнения). В качестве программных переменных мы будем использовать p с тем или иным целым индексом. Шагом про-

граммы является некоторое присваивание. Подготовительный раздел программы содержит присваивания

$$p_{-n-1} := a_n; \quad \ldots; \quad p_{-1} := a_0; \quad p_0 := x \tag{D.2}$$

и, возможно, ряд присваиваний вида $p_k := \ldots$, $k < -n-1$, с конкретными числами в правой части (т. е. можно запасти константы, нужные в вычислениях). Вычислительный раздел программы состоит из присваиваний вида

$$p_i := p_k, \quad k < i, \tag{D.3}$$

и

$$p_i := p_k \diamond p_l, \quad k, l < i, \quad \diamond \in \{+, \cdot\}. \tag{D.4}$$

Значение индекса в левой части будем считать номером шага, предполагая, что все используемые в программе значения индекса заполняют целиком некоторый отрезок множества целых чисел. Шаг вида (D.4) будем называть *аддитивным* или *мультипликативным* в зависимости от вида $\diamond$ ($+$ или $\cdot$). Шаг вида (D.3) будем называть *нейтральным*. Каждую программу такого вида, вычисляющую значение (D.1) и обладающую тем свойством, что, не меняя ее вычислительного раздела, а лишь варьируя правые части в (D.2), мы можем получать значения любых полиномов степени n в любых точках, мы назовем *n-программой*.

Наша цель состоит в доказательстве того, что каким бы ни было неотрицательное целое n, любая n-программа содержит не менее n аддитивных и не менее n мультипликативных шагов.

Если в n-программе изменить шаги с номерами $0, -1, \ldots, -n-1$, подставляя в правые части присваиваний (D.2) вместо чисел $x, a_0, a_1, \ldots, a_n$ их буквенные обозначения, и интерпретировать $+$, $\cdot$ в вычислительном разделе как знаки полиномиальных операций, то в результате выполнения n-программы значениями $p_1, p_2, \ldots$ будут полиномы (т. е. буквенные выражения, «картинки»). В частности, будет построен полином $a_n x^n + \ldots + a_1 x + a_0$, причем $a_n, \ldots, a_1, a_0$ и x являются переменными этого полинома (степень по x которого равна n, а степень, например, по a_n равна 1). При таком подходе каждая n-программа вычисления значения полинома превращается в программу построения полинома

$$P(x, a_0, a_1, \ldots, a_n) = a_n x^n + \ldots + a_1 x + a_0 \in \mathbb{Q}[x, a_0, a_1, \ldots, a_n] \tag{D.5}$$

с помощью операций сложения и умножения, исходя из переменных $x, a_0, a_1, \ldots, a_n$. Программу обсуждаемого вида, содержащую в под-

готовительном разделе формализованные указанным способом шаги с номерами $-n-1, -n, \ldots, 0$, назовем *формальной n-программой*. Если мы докажем, что любая формальная n-программа содержит не менее n аддитивных и не менее n мультипликативных шагов, то поставленная цель будет достигнута.

Лемма D.1. *Пусть $n \geqslant 0$, S — формальная n-программа, t — положительное целое, не превосходящее наибольшего значения индекса переменной p в S. Пусть среди шагов S с номерами $1, 2, \ldots, t$ нет аддитивных, в которых по крайней мере один операнд правой части зависит от a_n (имеет положительную степень по a_n как полином от $x, a_0, a_1, \ldots, a_n$). Тогда после выполнения S каждый из полиномов $p_1, p_2, \ldots, p_t$ либо не зависит от a_n, либо кратен a_n (может быть записан в виде $a_n Q$, где $Q \in \mathbb{Q}[x, a_0, a_1, \ldots, a_n]$).*

Доказательство. Индукция по t.

Для $t = 1$ утверждение очевидно.

Пусть $t > 1$ и утверждение леммы верно для $1, 2, \ldots, t-1$. Для шага с номером t имеется три возможности:

$$p_t := p_k, \quad p_t := p_k + p_l, \quad p_t := p_k \cdot p_l,$$

$k, l \leqslant t-1$. При осуществлении первой возможности требуемое непосредственно следует из предположения индукции. При второй возможности из сказанного в условии леммы относительно аддитивных шагов следует, что p_t не зависит от a_n. При третьей возможности любой из полиномов p_k, p_l может быть либо кратным a_n, либо не зависеть от a_n; в обоих случаях получаем то, что нам нужно. □

Лемма D.2. *Каждая формальная n-программа S, $n \geqslant 0$, построения полинома P вида* (D.5) *содержит не менее n аддитивных шагов.*

Доказательство. Индукция по n.

Для $n = 0$ утверждение очевидно.

Пусть $n > 0$ и утверждение леммы верно для $0, 1, \ldots, n-1$. Предположим, что S содержит менее n аддитивных шагов. Ясно, что в S имеется по крайней мере один аддитивный шаг такой, что по крайней мере один из его операндов зависит от a_n. В силу леммы D.1 первый такой шаг будет иметь по крайней мере один операнд, являющийся кратным a_n полиномом. Пусть этот шаг имеет вид $p_i := p_k + p_l$, и значением p_k (именно этот операнд мы берем только для определенности) является полином $a_n Q$, $Q \in \mathbb{Q}[x, a_0, a_1, \ldots, a_n]$. Если в S заменить шаг $p_{-n-1} := a_n$ на $p_{-n-1} := 0$, а шаг $p_i := p_k + p_l$ — на $p_i := p_l$, то, очевидно, получится формальная $(n-1)$-программа построения полино-

ма $a_{n-1}x^{n-1}+\ldots+a_1x+a_0$, содержащая менее чем $n-1$ аддитивных шагов. Противоречие. □

Исследуя число мультипликативных шагов, мы докажем утверждение более сильное, чем то, которое непосредственно нас интересует.

Во-первых, мы будем рассматривать формальные n-программы, которые строят полином, возможно лишь «по модулю x^{n+1}» равный $P(x,a_0,a_1,\ldots,a_n)$, т. е. некоторый полином вида $Ux^{n+1}+a_nx^n+\ldots$ $\ldots+a_1x+a_0$, где $U\in\mathbb{Q}[x,a_0,a_1,\ldots,a_n]$, и при этом конкретный вид U может быть любым, и он нас не будет интересовать. Мы согласны, таким образом, получить любой полином $W(x,a_0,a_1,\ldots,a_n)$ такой, что разность $W(x,a_0,a_1,\ldots,a_n)-P(x,a_0,a_1,\ldots,a_n)$ делится на x^{n+1}:

$$W(x,a_0,a_1,\ldots,a_n)\equiv P(x,a_0,a_1,\ldots,a_n)\pmod{x^{n+1}} \qquad \text{(D.6)}$$

(этим мы фактически расширяем понятие формальной n-программы). Во-вторых, мы будем исследовать число *существенно мультипликативных* шагов, т. е. таких шагов вида $p_i:=p_k\cdot p_l$, для которых $k,l\geqslant -n-1$; последнее означает, что операнды таких шагов не являются заранее запасенными константами. Ясно, что если для построения любого полинома $W(x,a_0,a_1,\ldots,a_n)$, удовлетворяющего (D.6), не существует формальной n-программы, содержащей менее n существенно мультипликативных шагов, то, в частности, не существует и формальной n-программы для построения $P(x,a_0,a_1,\ldots,a_n)$, содержащей менее n мультипликативных шагов.

Лемма D.3. *Пусть $n\geqslant 0$, S — формальная n-программа, t — положительное целое, не превосходящее наибольшего значения индекса переменной p в S. Пусть среди шагов S с номерами $1,2,\ldots,t$ нет существенно мультипликативных, в которых по крайней мере один операнд правой части зависит от a_n. Тогда после выполнения S каждый из полиномов $p_1,p_2,\ldots,p_t$ либо не зависит от a_n, либо имеет вид ca_n+Q, где $c\in\mathbb{Q}\setminus\{0\}$ и полином Q не зависит от a_n.*

Доказательство аналогично доказательству леммы D.1. □

Лемма D.4. *Каждая формальная n-программа S построения какого-либо W, удовлетворяющего* (D.6) *при $P=a_nx^n+\ldots+a_1x+a_0$, содержит не менее n существенно мультипликативных шагов.*

Доказательство. Индукция по n.

Для $n=0$ утверждение очевидно.

Пусть $n>0$ и утверждение леммы верно для $0,1,\ldots,n-1$. Пусть S — формальная n-программа построения некоторого полинома

$W(x, a_0, a_1, \ldots a_n)$, удовлетворяющего (D.6). Предположим, что в S меньше чем n существенно мультипликативных шагов. Покажем, что в таком случае можно построить формальную $(n-1)$-программу, в которой меньше чем $n-1$ существенно мультипликативных шагов, и тем самым получить противоречие с предположением индукции.

В силу леммы D.3 формальная n-программа S содержит по крайней мере один существенно мультипликативный шаг, по крайней мере один из операндов которого зависит от a_n. Пусть

$$p_i := p_k \cdot p_l \tag{D.7}$$

будет самым первым таким шагом, и пусть значением p_k является полином вида $ca_n + Q(x, a_0, a_1, \ldots, a_{n-1})$, $c \neq 0$. Тогда удаление из S всех шагов с номерами, большими $i-1$, и замена шага $p_{-n-1} := a_n$ шагом $p_{-n-1} := 0$ дает программу S', получающую полином $Q(x, a_0, a_1, \ldots, a_{n-1})$ в качестве значения переменной p_k. Пусть m — наименьшее значение индекса переменной p, встречающееся в S'. Добавим к S' шаг $p_{m-1} := c'$, где $c' = -1/c$, и шаг $p_i := p_{m-1} \cdot p_k$, который, заметим, не является существенно мультипликативным. Получаемая программа строит полином $-\frac{1}{c}Q(x, a_0, a_1, \ldots, a_{n-1})$; существенно мультипликативных шагов в ней на единицу меньше, чем среди шагов с номерами $1, 2, \ldots, i$ в S. Обозначим эту программу через S''.

Дальнейшие преобразования программы S имеют целью получение такой программы, которая содержит не более $n-1$ существенно мультипликативных шагов и при этом находит некоторый полином, по модулю x^n равный $a_{n-1}x^{n-1} + \ldots + a_1x + a_0$. Мы видим, что если бы значением p_{-n-1} было не a_n, а тот полином, который строится с помощью S'', то i-й шаг S привел бы к присваиванию переменной p_i значения 0, а после окончания выполнения всех шагов мы бы получили

$$W' = W(x, a_0, a_1, \ldots, a_{n-1}, c'Q(x, a_0, a_1, \ldots, a_{n-1})).$$

В силу (D.6) подстановка $a_n = c'Q(x, a_0, a_1, \ldots, a_{n-1})$ в W даст нам

$$W' = P(x, a_0, a_1, \ldots, a_{n-1}, c'Q(x, a_0, a_1, \ldots, a_{n-1})) + \\ + U'(x, a_0, a_1, \ldots, a_{n-1})x^{n+1}.$$

Принимая во внимание равенство $P = a_nx^n + \ldots + a_1x + a_0$, получаем $W' \in \mathbb{Q}[x, a_0, a_1, \ldots a_{n-1}]$ и

$$W' \equiv a_{n-1}x^{n-1} + \ldots + a_1x + a_0 \pmod{x^n}.$$

Отбросим предварительный раздел формальной n-программы S и запишем шаги ее вычислительного раздела вслед за S'', преобразуя

эти шаги следующим образом:

(a) если шаг не является шагом (D.7) или существенно мультипликативным шагом вида $p_t := p_r \cdot p_s$, $t \leqslant k$, то увеличиваем на i индекс в левой части и увеличиваем на i каждый из положительных индексов в правой части (неположительные индексы не изменяются);

(b) в правых частях всюду заменяем p_{-n-1} на p_i;

(c) каждый существенно мультипликативный шаг вида $p_t := p_r \cdot p_s$, $t \leqslant k$, заменяем нейтральным шагом $p_{t+i} := p_t$;

(d) шаг (D.7) заменяем нейтральным шагом $p_{2i} := p_{-n-1}$.

Прибегая к замене (c), мы пользуемся независимостью от a_n полиномов, являющихся значениями p_r и p_s, это позволяет не дублировать существенно мультипликативные шаги, содержащиеся в S''; завершающая замена (d) приводит нас к формальной $(n-1)$-программе построения W', имеющей менее $n-1$ существенно мультипликативных шагов. □

В силу сказанного ранее, из лемм D.2, D.4 следует теорема D.1[1].

Заметим, что замена каждого вычитания сложением с дополнительным домножением второго операнда на -1 не меняет числа существенно мультипликативных шагов. Поэтому доказательство теоремы D.1 сохраняет силу при рассмотрении сложения и вычитания в качестве аддитивных операций.

Известен алгоритм, который для вычисления значения произвольного полинома n-й степени требует n сложений и n умножений, это алгоритм («схема») Горнера:

$$a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x + a_0 = (\ldots (a_n x + a_{n-1}) x + \ldots + a_1) x + a_0.$$

Таким образом, мы получили следующий результат:

Если рассматривать n в качестве размера входа

$$x, a_0, a_1, \ldots, a_n,$$

то как по числу сложений, так и по числу умножений схема Горнера является оптимальным в худшем случае алгоритмом вычисления значения $a_n x^n + \ldots + a_1 x + a_0$ с помощью операций сложения и умножения.

Разумеется, полиномы какого-либо частного вида, как, например, x^n, могут вычисляться с меньшими затратами.

[1] В 1962 г. В. Я. Пан доказал утверждение теоремы D.1 в несколько более общей форме — см., например, [19, разд. 4.6.4, упр. 38]. Главные идеи доказательства, приведенного выше в этом приложении, взяты из [57].

Приложение Е

Об одном свойстве сумм неотрицательных случайных величин

Здесь доказывается теорема 17.1 из § 17.

Пусть $\xi_1, \xi_2, \ldots$ *— последовательность неотрицательных случайных величин на некотором вероятностном пространстве. Пусть числовая последовательность* $h_1, h_2, \ldots$ *такова, что для каждого* $k \geqslant 1$ *выполнено* $\mathsf{E}(\xi_k | \xi_1, \xi_2, \ldots, \xi_{k-1}) \leqslant h_k$ *при всех значениях* $\xi_1, \xi_2, \ldots, \xi_{k-1}$. *Зафиксируем неотрицательное число* q *и введем целочисленную случайную величину*

$$\tau = \min\Big\{n\colon \sum_{k=1}^{n} \xi_k \geqslant q\Big\}.$$

Пусть $\tau < \infty$ *всюду на рассматриваемом вероятностном пространстве. Тогда* $\mathsf{E}\Big(\sum\limits_{k=1}^{\tau} h_k\Big) \geqslant q$.

Рассмотрим случайные величины χ_k, $k = 1, 2, \ldots$, где $\chi_1 = 1$, а при $k > 1$ выполняется $\chi_k = 1$, если $\xi_1 + \xi_2 + \ldots + \xi_{k-1} < q$, и $\chi_k = 0$ в противном случае. Заметим, что

$$\chi_k = \begin{cases} 1, & \text{если } \tau \geqslant k, \\ 0, & \text{если } \tau < k, \end{cases}$$

и $1 = \chi_1 \geqslant \chi_2 \geqslant \ldots$

Положим $p_k = \mathsf{P}(\tau = k) = \mathsf{P}(\chi_k = 1) - \mathsf{P}(\chi_{k+1} = 1)$, $k = 1, 2, \ldots$ Ясно, что $\mathsf{P}(\chi_k = 1) = 1 - p_1 - p_2 - \ldots - p_{k-1}$. Имеем

$$\mathsf{E}\Big(\sum_{k=1}^{\tau} h_k\Big) = \sum_{k=1}^{\infty} \sum_{j=1}^{k} h_j \mathsf{P}(\tau = k) =$$

$$= \sum_{k=1}^{\infty} (\mathsf{P}(\chi_k = 1) - \mathsf{P}(\chi_{k+1} = 1)) \sum_{j=1}^{k} h_j =$$

$$
\begin{aligned}
&= \sum_{k=1}^{\infty} \mathsf{P}(\chi_k = 1) \sum_{j=1}^{k} h_j - \sum_{k=1}^{\infty} \mathsf{P}(\chi_{k+1} = 1) \sum_{j=1}^{k} h_j = \\
&= \sum_{k=1}^{\infty} \mathsf{P}(\chi_k = 1) \sum_{j=1}^{k} h_j - \sum_{k=2}^{\infty} \mathsf{P}(\chi_k = 1) \sum_{j=1}^{k-1} h_j = \\
&= \sum_{k=1}^{\infty} \mathsf{P}(\chi_k = 1) \sum_{j=1}^{k} h_j - \sum_{k=2}^{\infty} \mathsf{P}(\chi_k = 1) \Big(\sum_{j=1}^{k} h_j - h_k \Big) = \\
&= \sum_{k=1}^{\infty} \mathsf{P}(\chi_k = 1) \sum_{j=1}^{k} h_j - \sum_{k=2}^{\infty} \mathsf{P}(\chi_k = 1) \sum_{j=1}^{k} h_j + \sum_{k=2}^{\infty} \mathsf{P}(\chi_k = 1) h_k = \\
&= h_1 \mathsf{P}(\chi_1 = 1) + \sum_{k=2}^{\infty} h_k \mathsf{P}(\chi_k = 1) = \\
&= \sum_{k=1}^{\infty} h_k \mathsf{P}(\chi_k = 1).
\end{aligned}
$$

Таким образом,

$$
\mathsf{E}\Big(\sum_{k=1}^{\tau} h_k \Big) = \sum_{k=1}^{\infty} h_k \mathsf{P}(\chi_k = 1). \tag{E.1}
$$

Введем случайные величины $\eta_k = \chi_k \xi_k$, $k = 1, 2, \ldots$; из определения случайных величин χ_k следует, что

$$
\sum_{k=1}^{\infty} \eta_k \geqslant q. \tag{E.2}
$$

Докажем, что для $k = 1, 2, \ldots$

$$
\mathsf{E}\eta_k \leqslant h_k \mathsf{P}(\chi_k = 1). \tag{E.3}
$$

Рассмотрим полную группу событий W_1, W_2: в W_1 попадают те элементы вероятностного пространства, для которых $\xi_1 + \xi_2 + \ldots + \xi_{k-1} < q$, в W_2 — для которых $\xi_1 + \xi_2 + \ldots + \xi_{k-1} \geqslant q$. С учетом определения случайной величины χ_k, равной 1 на множестве W_1 и 0 на множестве W_2, получаем по формуле полного математического ожидания:

$$
\begin{aligned}
\mathsf{E}\eta_k &= \mathsf{E}(\chi_k \xi_k) = \\
&= \mathsf{E}(\chi_k \xi_k | W_1) \mathsf{P}(W_1) + \mathsf{E}(\chi_k \xi_k | W_2) \mathsf{P}(W_2) = \\
&= \mathsf{E}(\xi_k | W_1) \mathsf{P}(W_1) = \\
&= \mathsf{E}(\xi_k | W_1) \mathsf{P}(\chi_k = 1),
\end{aligned}
$$

т. е.

$$\mathsf{E}\eta_k = \mathsf{E}(\xi_k|W_1)\mathsf{P}(\chi_k = 1). \tag{E.4}$$

Так как по условию неравенство $\mathsf{E}(\xi_k|\xi_1, \xi_2, \ldots, \xi_{k-1}) \leqslant h_k$ выполняется при всех значениях $\xi_1, \xi_2, \ldots, \xi_{k-1}$, то оно выполняется на W_1. С учетом (E.4) имеем $\mathsf{E}(\xi_k|W_1)\mathsf{P}(\chi_k = 1) \leqslant h_k$, откуда следует (E.3).

Из (E.1), (E.2), (E.3) получаем

$$q = \mathsf{E}q \leqslant \mathsf{E}\Big(\sum_{k=1}^{\infty} \eta_k\Big) = \sum_{k=1}^{\infty} \mathsf{E}\eta_k \leqslant \sum_{k=1}^{\infty} h_k \mathsf{P}(\chi_k = 1) = \mathsf{E}\Big(\sum_{k=1}^{\tau} h_k\Big),$$

что и требовалось[1].

[1] Сходное, но менее подробное доказательство имеется в [2, гл. III, § 5, теорема 1], но там в условии теоремы пропущено требование конечности τ (см. задачу 96).

Приложение F

О сортировке, оптимальной по числу сравнений в худшем случае

Долгое время считалось правдоподобным предположение, что сортировкой, оптимальной по числу сравнений в худшем случае, является сортировка бинарными вставками, о сложности $T_B(n)$ которой, вместе с этим, было известно, что $T_B(5) = 8$, при том, что наибольшая из обоснованных нижних границ для числа сравнений, необходимых для сортировки пяти элементов, есть $\lceil \log_2 5! \rceil = 7$. Это порождало гипотезу, что сложность $T_{opt}(n)$ оптимальной сортировки для некоторых значений n больше, чем $\lceil \log_2 n! \rceil$, и первое такое значение n равно пяти. Но в 1956 г. Г. Б. Демутом был найден алгоритм сортировки пяти элементов, который требует всего семь сравнений.

Этот алгоритм можно легко изобразить с помощью рисунков, на которых те точки, которые соответствуют сравнивавшимся элементам, соединяются стрелкой, ведущей от большего элемента к меньшему. Сравниваем первый элемент со вторым и третий с четвертым, а потом сравниваем меньшие найденные элементы; на это уйдет три сравнения (рис. 28). Этим, в частности, для трех элементов из пяти мы устанавливаем их относительный порядок. Затем для пятого элемента, который еще не сравнивался, мы находим место среди трех уже упорядоченных элементов бинарным поиском; на это уйдет два сравнения. Тем самым мы приходим к одному из двух случаев, изображенных на рис. 29. И в том, и в другом случае для завершения сортировки с помощью бинарного поиска места элемента достаточно двух сравнений: в случае, изображенном на рис. 29а, вставка производится в массив из двух элементов, в случае, изображенном на рис. 29б, — из трех элементов. В итоге семи сравнений оказывается достаточно.

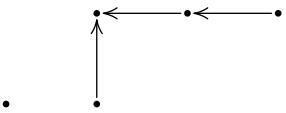


Рис. 28. Сортировка пяти элементов: ситуация после трех сравнений.

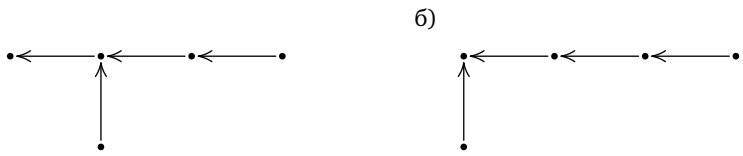


Рис. 29. Сортировка пяти элементов, ситуация после нахождения места пятого элемента среди трех упорядоченных элементов: а) пятый элемент меньше всех трех упорядоченных элементов; б) пятый элемент больше наименьшего из трех упорядоченных элементов. (В обоих случаях еще двух сравнений достаточно для завершения сортировки.)

Указанный алгоритм сортировки пяти элементов был позднее обобщен на произвольное число элементов, эта сортировка получила название сортировки слияниями и вставками. Через $T_{\mathrm{F}}(n)$ обычно обозначается сложность этой сортировки по числу сравнений. Обнаружено, что $T_{\mathrm{F}}(n) = \lceil \log_2 n! \rceil$ для $n = 1, 2, \ldots, 11$. Но при этом $T_{\mathrm{F}}(12) = 30$, хотя $\lceil \log_2 12! \rceil = 29$, и возникает вопрос, всегда ли возможна сортировка двенадцати элементов не более чем двадцатью девятью сравнениями? Аналогичные вопросы могут возникнуть и для бóльших значений n.

Как говорилось в § 14, имеется алгоритм, который, исходя из $n > 0$, строит оптимальный по числу сравнений алгоритм сортировки массивов из n элементов (алгоритм строит алгоритм). Этот алгоритм построения оптимального алгоритма сортировки требует огромной работы, даже если применить все средства экономии перебора. Но, тем не менее, на этом пути с помощью компьютера показано, что $T_{\mathrm{opt}}(12) = 30$, что больше, чем $\lceil \log_2 12! \rceil$. Но дальше продвинуться не удалось, и по сегодняшний день, видимо, неизвестно число $T_{\mathrm{opt}}(13)$:

n:	1	2	3	4	5	6	7	8	9	10	11	12	13
$\lceil \log_2 n! \rceil$:	0	1	3	5	7	10	13	16	19	22	26	29	33
$T_{\mathrm{B}}(n)$:	0	1	3	5	8	10	14	17	21	25	29	33	37
$T_{\mathrm{F}}(n)$:	0	1	3	5	7	10	13	16	19	22	26	30	34
$T_{\mathrm{opt}}(n)$:	0	1	3	5	7	10	13	16	19	22	26	30	?

Исследования, не связанные с компьютерным вычислением $T_{\mathrm{opt}}(n)$, показали, что имеется бесконечно много значений n, для которых $T_{\mathrm{F}}(n) > T_{\mathrm{opt}}(n)$. Значение 47 — наименьшее из известных[1].

[1] См. [20, разд. 5.3.1].

Приложение G

Метод построения общего решения линейного рекуррентного уравнения с постоянными коэффициентами

Напомним метод построения общего решения линейного рекуррентного уравнения с постоянными коэффициентами [1]. Начнем со случая однородного уравнения

$$a_d y(n) + a_{d-1} y(n-1) + \ldots + a_0 y(n-d) = 0. \tag{G.1}$$

Пусть $\lambda_1, \lambda_2, \ldots, \lambda_s$ — все различные корни *характеристического уравнения*

$$a_d \lambda^d + a_{d-1} \lambda^{d-1} + \ldots + a_0 = 0 \tag{G.2}$$

и $m_1, m_2, \ldots, m_s$ — кратности этих корней, $m_1 m_2 + \ldots + m_s = d$. Общим решением уравнения (G.1) является

$$\sum_{k=1}^{s} (C_{k,m_k-1} n^{m_k-1} + \ldots + C_{k1} n + C_{k0}) \lambda_k^n, \tag{G.3}$$

где все C_{ij} — произвольные постоянные.

Общее решение неоднородного линейного рекуррентного уравнения с постоянными коэффициентами

$$a_d y(n) + a_{d-1} y(n-1) + \ldots + a_0 y(n-d) = f(n), \tag{G.4}$$

где $f(n)$ — известная функция, является суммой общего решения соответствующего однородного уравнения (G.1) и какого-нибудь частного решения неоднородного уравнения (G.4).

Если правая часть $f(n)$ уравнения (G.4) представлена в виде $f(n) = g(n) + \ldots + h(n)$ и если известны частные решения $v(n), \ldots, w(n)$ для уравнений, левые части которых совпадают с левой частью уравнения (G.4), а правые части равны соответственно

[1] См., например, [12, гл. V, § 4], [32, гл. 3].

$g(n), \ldots, h(n)$, то $v(n) + \ldots + w(n)$ будет частным решением уравнения (G.4).

Если правая часть $f(n)$ неоднородного уравнения (G.4) представляет собой *квазиполином*, т. е. равна $p(n)\mu^n$, где $p(n)$ — полином некоторой степени $l \geqslant 0$, а μ — некоторое (комплексное) число, то (G.4) имеет частное решение в виде квазиполинома $q(n)\mu^n$, и при этом полином $q(n)$ имеет степень $l + m$, где m — кратность μ как корня характеристического уравнения (G.2) (если μ не является корнем этого уравнения, то считаем кратность нулевой: $m = 0$). Коэффициенты полинома $q(n)$ находятся методом неопределенных коэффициентов.

Частные решения с заданными начальными значениями находятся подстановкой в общее решение начальных значений и определением постоянных из возникшей системы линейных алгебраических уравнений. (С помощью этого метода выводится и формула Бине (10.2).)

Приложение H

Об одном семействе алгебраических уравнений

В примере 24.3 мы столкнулись с рекуррентным уравнением

$$y(n) - y(n-1) - \ldots - y(n-k) = 1, \tag{H.1}$$

$y(0) = y(1) = \ldots = y(k-1) = 0$. В качестве частного решения этого неоднородного уравнения при $k > 1$ можно взять $\frac{-1}{k-1}$. Нам предстоит теперь заняться общим решением соответствующего однородного уравнения, характеристическим уравнением которого служит

$$\lambda^k - \lambda^{k-1} - \ldots - \lambda - 1 = 0. \tag{H.2}$$

Свойства корней уравнения (H.2) описываются следующим предложением.

Предложение H.1. *При $k > 1$ уравнение* (H.2) *имеет k различных корней. При этом в точности один из них — мы будем называть его главным корнем уравнения* (H.2) *и обозначать через α_k — по модулю превосходит единицу. Главный корень — вещественное число, удовлетворяющее условию*[1] $2 - \frac{1}{k} \leqslant \alpha_k < 2$.

Доказательство. Поскольку предстоит использовать некоторую технику теории аналитических функций, мы перейдем к привычному обозначению z для комплексной переменной и будем рассматривать уравнение $z^k - z^{k-1} - \ldots - z - 1 = 0$, левую часть которого обозначим через $\nu_k(z)$. Положим $V_k(z) = (z-1)\nu_k(z) = z^{k+1} - 2z^k + 1$.

Уравнение $V_k(z) = 0$ в сравнении с $\nu_k(z) = 0$ имеет дополнительный корень $z = 1$; мы докажем утверждение предложения для $V_k(z) = 0$,

[1] См. [28, гл. 13, пп. 7—12], [54], [20, разд. 5.4.2, упр. 7]. В [28] доказано также, что все корни, отличные от главного корня α_k, по модулю не превосходят $\alpha_k - 1 < 1$.

$k>1$, откуда будет следовать требуемое. Доказательство мы проведем в три этапа, установив справедливость следующих утверждений:

(i) для любого $k>1$ все корни уравнения $V_k(x)=0$ простые (т. е. имеют кратность 1);

(ii) круг любого радиуса r, $1<r<2-\frac{1}{k}$, с центром в $z=0$ содержит ровно k корней уравнения $V_k(z)=0$;

(iii) уравнение $V_k(z)=0$ имеет по меньшей мере один вещественный корень на полуинтервале $\left[2-\frac{1}{k},2\right)$.

Для доказательства утверждения (i) заметим, прежде всего, что

$$\text{нод}(V_k(z),V_k'(z))=\text{нод}(V_k(z),z^{k-1}((k+1)z-2k)).$$

Но $V_k(z)$ не обращается в 0 при $z=0$ и при $z=\frac{2k}{k+1}$. Отсюда следует, что $\text{нод}(V_k(z),V_k'(z))=1$. Это говорит о том, что корни $V_k(z)$ простые.

Для доказательства утверждения (ii) положим $W_k(z)=z^{k+1}-2z^k$. Если на некоторой окружности выполняется неравенство $|W_k(z)|>1$, то внутри этой окружности полиномы $V_k(z)$ и $W_k(z)$ по теореме Руше [1] имеют одинаковое число корней. Рассмотрим окружность S_σ радиуса $1+\sigma$, $0<\sigma<1$, с центром в $z=0$. Так как

$$|W_k(z)|=|z^{k+1}-2z^k|\geqslant|2z^k|-|z^{k+1}|,$$

то на окружности S_σ выполняется $|W_k(z)|\geqslant(1+\sigma)^k(1-\sigma)$. Далее, очевидно, $(1+\sigma)^k\geqslant 1+k\sigma$, и мы получаем, что если σ удовлетворяет неравенству

$$(1+k\sigma)(1-\sigma)>1, \tag{Н.3}$$

то условия теоремы Руше выполнены, и полином $V_k(z)$ имеет внутри S_σ столько же корней, сколько их имеет $W_k(x)$, т. е. k. Неравенство (Н.3) выполнено для всех значений σ, принадлежащих интервалу с концами, являющимися корнями квадратного уравнения $k\sigma^2-(k-1)\sigma=0$, — эти корни суть 0 и $1-\frac{1}{k}$. Это доказывает утверждение (ii).

Наконец, утверждение (iii) следует из того, что $\nu_k(1)<0$, $\nu_k(2)>0$ и при этом уравнение $\nu_k(z)=0$ не имеет корней на интервале $\left(1,2-\frac{1}{k}\right)$. □

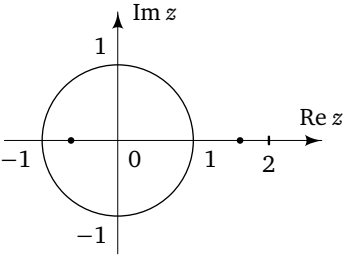
[1] См., например, [34, гл. 5, § 3] или [29, п. 1.1]. Применительно к полиномиальному случаю эта теорема допускает следующую формулировку. Пусть полином $V(z)$ представлен в виде суммы $W(z)+\widetilde{W}(z)$ двух полиномов, и на контуре некоторой области значение $|\widetilde{W}(z)|$ меньше значения $|W(z)|$. Тогда внутри этой области число корней полиномов $V(z)$ и $W(z)$ одинаково.

Рекуррентное уравнение (Н.1) имеет, таким образом, общее решение

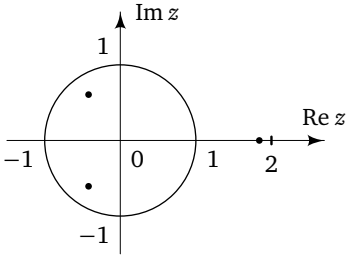
$$y(n) = C_k\alpha_k^n + C_{k-1}\alpha_{k-1}^n + \ldots + C_1\alpha_1^n - \frac{1}{k-1}$$

(α_k — главный корень характеристического уравнения (Н.2), все остальные корни $\alpha_1, \alpha_2, \ldots, \alpha_{k-1}$ по модулю не превосходят единицу). Нас интересует частное решение, удовлетворяющее условию $y(0) = y(1) = \ldots = y(k-1) = 0$. И без нахождения всех C_i ясно, что $C_k \neq 0$, — иначе последовательность $y(k), y(k+1), \ldots$ была бы ограниченной.

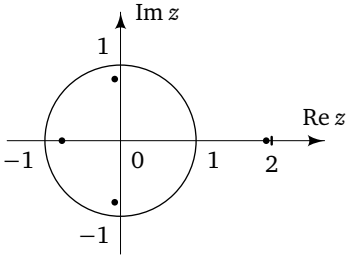
На рис. 30 показано расположение корней уравнений $V_k(z) = 0$, $k = 2, 3, 4$.



а) $z^2 - z - 1 = 0$



б) $z^3 - z^2 - z - 1 = 0$



в) $z^4 - z^3 - z^2 - z - 1 = 0$

Рис. 30.

Итак, нам удалось обойтись без фактического нахождения корней характеристического уравнения. Более того, речь шла не об одном конкретном уравнении, а о семействе, в которое входят уравнения сколь угодно высоких степеней.

Мы получили доказательство предложения 24.1 из § 24.

Литература

[1] *С. А. Абрамов*. Исследование алгоритмов одновременного нахождения наибольшего и наименьшего элементов массива // ЖВМ и МФ. 1982. Т. 22, № 2. С. 424—428.

[2] *С. А. Абрамов*. Элементы анализа программ. Частичные функции на множестве состояний. М.: Наука, 1986.

[3] *С. А. Абрамов, Г. Г. Гнездилова*. Алгоритм управления вопросником в автоматизированной обучающей системе // Вестн. Моск. ун-та. Сер. 15. Вычисл. мат. и киб. 1988. № 2. С. 47—52.

[4] *В. Б. Алексеев*. Введение в теорию сложности алгоритмов: Учебное пособие для студентов. М.: Издательский отдел ф-та ВМиК МГУ, 2002.

[5] *А. Ахо, Дж. Хопкрофт, Дж. Ульман*. Построение и анализ вычислительных алгоритмов. М.: Мир, 1979.

[6] *Н. Г. де Брейн*. Асимптотические методы в анализе. М.: ИЛ, 1961.

[7] *А. А. Васин, В. В. Морозов*. Теория игр и модели математической экономики. Учебное пособие. М.: МАКС Пресс, 2005.

[8] Введение в криптографию / Под общей редакцией В. В. Ященко. М.: МЦНМО: ЧеРо, 2000.

[9] *Н. К. Верещагин, А. Шень*. Начала теории множеств. М.: МЦНМО, 2008.

[10] *М. Н. Вялый*. Сложность вычислительных задач // Математическое просвещение, вып. 4. М.: МЦНМО, 2000. С. 81—114.

[11] *С. Б. Гашков, В. Н. Чубариков*. Арифметика. Алгоритмы. Сложность вычислений. М.: Высшая школа, 2000.

[12] *А. О. Гельфонд*. Исчисление конечных разностей. М.: Наука, 1967.

[13] *М. Гэри*, *Д. Джонсон*. Вычислительные машины и труднорешаемые задачи. М.: Мир, 1982.

[14] *В. А. Ильин*, *Г. Д. Ким*. Линейная алгебра и аналитическая геометрия. М.: Изд-во Моск. ун-та, 1998.

[15] *А. А. Карацуба*, *Ю. П. Офман*. Умножение многозначных чисел на автоматах // Докл. АН СССР. 1962. Т. 145, № 2. С. 293—294.

[16] *А. А. Карацуба*. Основы аналитической теории чисел. М.: Наука, 1975.

[17] *А. А. Карацуба*. Сложность вычислений // Труды Математического института РАН. 1995. Т. 211. С. 186—203.

[18] *Д. Кнут*. Искусство программирования для ЭВМ. Т. 1. Основные алгоритмы. М.—СПб.—Киев: ИД «Вильямс», 1999.

[19] *Д. Кнут*. Искусство программирования для ЭВМ. Т. 2. Получисленные алгоритмы. М.—СПб.—Киев: ИД «Вильямс», 2000.

[20] *Д. Кнут*. Искусство программирования для ЭВМ. Т. 3. Сортировка и поиск. М.—СПб.—Киев: ИД «Вильямс», 2001.

[21] *Т. Кормен*, *Ч. Лейзерсон*, *Р. Ривест*. Алгоритмы: построение и анализ. М.: МЦНМО, 2000.

[22] *А. И. Кострикин*. Введение в алгебру. М.: Наука, 1977.

[23] *В. Н. Крупский*. Введение в сложность вычислений. М.: Факториал-Пресс, 2006.

[24] *С. С. Лавров*. Программирование. Математические основы, средства, теория. СПб: БХВ-Петербург, 2001.

[25] *В. Н. Латышев*. Комбинаторная теория колец: сложность алгебраических алгоритмов. М.: Изд-во Моск. ун-та, 1987.

[26] *Дж. Макконелл*. Анализ алгоритмов. Вводный курс. М.: Техносфера, 2002.

[27] *Ф. Мостеллер*. 50 занимательных вероятностных задач с решениями. М.: URSS, 2005.

[28] *А. М. Островский*. Решение уравнений и систем уравнений. М.: ИЛ, 1963.

[29] *В. В. Прасолов*. Многочлены. М.: МЦНМО, 2003.

[30] *Ф. Препарата*, *М. Шеймос*. Вычислительная геометрия: введение. М.: Мир, 1989.

[31] *Э. Рейнгольд*, *Ю. Нивергельт*, *Н. Део*. Комбинаторные алгоритмы. Теория и практика. М.: Мир, 1980.

[32] *В. К. Романко*. Разностные уравнения. М.: БИНОМ. Лаборатория знаний, 2006.

[33] *А. А. Сапоженко*. Некоторые вопросы сложности алгоритмов: учебное пособие. М.: Издательский отдел ф-та ВМиК МГУ, 2001.

[34] *А. Г. Свешников*, *А. Н. Тихонов*. Теория функций комплексной переменной. М.: Физматлит, 1999.

[35] *В. Серпинский*. 250 задач по теории чисел. Москва—Ижевск: Регулярная и хаотическая динамика, 2004.

[36] *А. Л. Тоом*. О сложности схемы из функциональных элементов, реализующей умножение целых чисел // Докл. АН СССР. 1963. Т. 150, № 2. С. 496—498.

[37] *Дж. Хартманис*, *Дж. Э. Хопкрофт*. Обзор теории сложности // Кибернетический сборник. 1974. Вып. 11. С. 131—176.

[38] *П. Л. Чебышев*. Полное собрание сочинений. Т. 1. М.—Л.: АН СССР, 1944.

[39] *И. Р. Шафаревич*. Избранные главы алгебры. М.: Журнал «Математическое образование», 2000.

[40] Математическая энциклопедия. Т. 1. М.: Издательство «Советская энциклопедия», 1977.

[41] *S. A. Abramov*, *M. Bronstein*. Hypergeometric dispersion and the orbit problem // Proceedings of the 2000 International Symposium on Symbolic and Algebraic Computation. New York: ACM, 2000. P. 8—13.

[42] *M. Agrawal*, *N. Kayal*, *N. Saxena*. PRIMES is in P // Ann. of Math. (2). 2004. V. 160, № 2. P. 781—793.

[43] *S. Baase*, *A. van Gelder*. Computer algorithms: introduction to design and analysis. Boston, MA: Addison-Wesley, 2000.

[44] *E. Bach*, *J. Shallit*. Algorithmic number theory. V. 1. Efficient algorithms. Cambridge, MA: MIT Press, 1996.

[45] *P. E. Blanksby*, *H. L. Montgomery*. Algebraic integers near the unit circle // Acta Arith. 1971. V. 18. P. 355—369.

[46] *G. Brassard*, *P. Bratley*. Fundamentals of algorithms. Englewood Cliffs, NJ: Prentice Hall, 1996.

[47] *D. Coppersmith*, *S. Winograd*. Matrix multiplication via arithmetic progressions // J. Symbolic Comput. 1990. V. 9, № 3. P. 251—280.

[48] *M. J. Fischer*, *M. O. Rabin*. Super-exponential complexity of Presburger arithmetic // Complexity of computation (Proc. SIAM-AMS Sympos., New York, 1973). Providence, RI: AMS, 1974. (SIAM-AMS Proc.; Vol. VII). P. 27—41.

[49] *N. Friedman*. Some results on the effect of arithmetics on comparison problems // Proc. 13th IEEE Ann. Symp. on Switching and Automata Theory. Los Alamitos, CA: IEEE Computer Society, 1972. P. 139—143.

[50] *J. von zur Gathen*, *J. Gerhard*. Modern computer algebra. New York: Cambrige University Press, 1999.

[51] *R. Kannan*, *R. J. Lipton*. Polynomial-time algorithm for the orbit problem // J. Assoc. Comput. Mach. 1986. V. 33, № 4. P. 808—821.

[52] *D. E. Knuth*. Big omicron and big omega and big theta // ACM SIGACT News. 1976. V. 8, iss. 2. P. 18—23.

[53] *J. C. Lagarias.* The $3x + 1$ problem and its generalizations // The American Mathematical Monthly. 1989. V. 92, № 1. P. 3—23.

[54] *E. P. Miles, Jr.* Generalized Fibonacci numbers and associated matrices // The American Mathematical Monthly. 1960. V. 67, № 8. P. 745—752.

[55] *R. Motwani*, *P. Raghavan*. Randomized algorithms. Cambridge: Cambridge University Press, 1995.

[56] *I. Pohl*. A sorting problem and its complexity // Comm. ACM. 1972. V. 15, № 6. P. 462—466.

[57] *E. M. Reingold*, *A. I. Stocks*. Simple proofs of lower bounds for polynomial evaluation // Complexity of computer computations (Proc. Sympos., IBM Thomas J. Watson Res. Center, Yorktown Heights, N. Y., 1972). New York: Plenum, 1972. P. 21—30.

[58] *A. Schinzel*, *H. Zassenhaus*. A refinement of two theorems of Kronecker // Michigan Math. J. 1965. V. 12. P. 81—85.

[59] *R. Sedgewick*, *P. Flajolet*. An introduction to the analysis of algorithms. Boston, MA: Addison-Wesley, 1996.

Предметный указатель

Оглавление

Сергей Александрович Абрамов

Лекции о сложности алгоритмов

Издательство Московского центра
непрерывного математического образования
119002, Москва, Большой Власьевский пер., 11. Тел. (499) 241-74-83

Подписано в печать 01.11.2008 г. Формат $60\times90\,1/16$. Бумага офсетная.
Печать офсетная. Печ. л. 16. Тираж 1000. Заказ .

Отпечатано с готовых диапозитивов в ППП «Типография „Наука“».
121099, Москва, Шубинский пер., д. 6.

Книги издательства МЦНМО можно приобрести в магазине «Математическая книга»,
Большой Власьевский пер., д. 11. Тел. (499) 241-72-85. E-mail: biblio@mccme.ru