А. Ахо

Построение и анализ вычислительных алгоритмов

Москва
«Книга по Требованию»

УДК 51
ББК 22.1
А11

А. Ахо
А11 Построение и анализ вычислительных алгоритмов / А. Ахо – М.: Книга по Требованию, 2013. – 534 с.

ISBN 978-5-458-26604-8

Классика Computer Science - книга Ахо, Ульмана и Хопкрофта. Шаблоны постороения эффективных алгоритмов, рассмотрены алгоритмы сортировки, порядковых статистик, представление множеств, алгоритмы на графах, операции с матрицами, алгоритмы теории чисел, идентификация (регулярные выражения) и тд. Книга не устареет никогда!

ISBN 978-5-458-26604-8

ПРЕДИСЛОВИЕ

Изучение алгоритмов является самой сердцевиной науки о вычислениях. В последние годы здесь были достигнуты значительные успехи. Они простираются от разработки более быстрых алгоритмов, таких, как быстрое преобразование Фурье, до впечатляющего открытия, что для некоторых естественных проблем все алгоритмы неэффективны. Эти результаты вызвали громадный интерес к изучению алгоритмов, и их стали интенсивно разрабатывать и исследовать. Цель данной книги — собрать вместе существенные результаты в этой области, чтобы облегчить понимание принципов и концепций, на которых зиждется разработка алгоритмов.

Содержание книги

Для анализа работы алгоритма нужна какая-нибудь модель вычислительной машины. Наша книга начинается с определения нескольких таких моделей, достаточно простых для анализа, но в то же время точно отражающих основные черты реальных машин. Эти модели включают машину с произвольным доступом к памяти, машину с произвольным доступом к памяти и хранимой программой, а также некоторые их разновидности. Машина Тьюринга вводится для доказательства экспоненциальных нижних оценок эффективности алгоритмов в гл. 10 и 11. Поскольку общая тенденция в разработке программ состоит в отходе от использования машинно-ориентированных языков, вводится язык высокого уровня, называемый Упрощенным Алголом (Pidgin ALGOL), как основное средство для описания алгоритмов. Сложность программы на Упрощенном Алголе связывается с соответствующей моделью машины.

В гл. 2 вводятся основные структуры данных и техника программирования, часто применяемые в эффективных алгоритмах. Она охватывает списки, магазины, очереди, деревья и графы. Приведено подробное объяснение рекурсии, приема "разделяй и властвуй" и динамического программирования вместе с примерами их применения.

В гл. 3—9 указаны разнообразные области, в которых может применяться основополагающая техника гл. 2. В этих главах мы уделяем главное внимание построению алгоритмов, асимптотически наилучших среди известных. По этой причине некоторые из рассматриваемых алгоритмов хороши лишь для входных данных, длина которых много больше, чем у обычно встречающихся на практике. В частности, это относится к некоторым алгоритмам умножения матриц из гл. 6, к принадлежащему Шёнхаге и Штрассену алгоритму умножения целых чисел из гл. 7 и некоторым алгоритмам из гл. 8, работающим с полиномами и целыми числами.

С другой стороны, бóльшая часть алгоритмов сортировки из гл. 3, поиска из гл. 4, алгоритмов на графах из гл. 5, быстрое преобразование Фурье из гл. 7 и алгоритмы идентификации подслов из гл. 9 используются широко, поскольку размеры входов, на которых эти алгоритмы эффективны, достаточно малы, чтобы встретить их во многих практических ситуациях.

В гл. 10—12 обсуждаются нижние границы сложности вычислений. Подлинная вычислительная сложность задачи интересна вообще — и для разработки программ, и для понимания природы вычисления. В гл. 10 изучается важный класс задач — так называемые NP-полные задачи. Все задачи из этого класса эквивалентны по вычислительной сложности в том смысле, что если одна из них имеет эффективное (с полиномиально ограниченным временем) решение, то все они имеют эффективные решения. Так как этот класс содержит много практически важных и долго изучавшихся задач, таких, как задача целочисленного программирования или задача о коммивояжере, то есть основания подозревать, что ни одну из задач этого класса нельзя решить эффективно. Поэтому если разработчик программы знает, что задача, для которой он пытается найти эффективный алгоритм, принадлежит этому классу, то ему, возможно, надо удовлетвориться попытками эвристического подхода к ней. Несмотря на огромное число эмпирических свидетельств в пользу противоположного, до сих пор открыт вопрос, допускают ли NP-полные задачи эффективные решения.

В гл. 11 описаны конкретные задачи, для которых можно доказать, что эффективных алгоритмов для них действительно нет. Материал гл. 10 и 11 существенно опирается на понятие машины Тьюринга, введенное в разд. 1.6 и 1.7.

В последней главе мы устанавливаем связь трудности вычисления с понятием линейной независимости в векторных пространствах. Материал этой главы дает технику доказательства нижних оценок для гораздо более простых задач, чем рассмотренные в гл. 10 и 11.

О пользовании книгой

Эта книга задумана как начальный курс по разработке и анализу алгоритмов. Основной упор сделан на идеи и простоту понимания, а не на проработку деталей реализации или программистские трюки. Часто вместо длинных утомительных доказательств даются лишь неформальные интуитивные объяснения.

Книга замкнута в себе и не предполагает специальной подготовки ни по математике, ни по языкам программирования. Однако желательна определенная зрелость в навыках обращения с математическими понятиями, а также некоторое знакомство с языками программирования высокого уровня, такими, как Фортран или Алгол. Для полного понимания гл. 6—8 и 12 нужны некоторые познания в линейной алгебре.

Эта книга использовалась как основа для спецкурсов по разработке алгоритмов. За один семестр изучался материал, покрывающий бо́льшую часть гл. 1—5, 9 и 10 вместе с беглым обзором тематики остальных глав. Во вводных курсах основное внимание уделялось материалу из гл. 1—5, но разд. 1.6, 1.7, 4.13, 5.11 и теорема 4.5 обычно не изучались. Книгу можно также использовать для более глубоких спецкурсов, делая упор на теорию алгоритмов. Основой для этого могли бы служить гл. 6—12.

В конце каждой главы приведены упражнения, обеспечивающие преподавателя широким выбором домашних заданий. Упражнения классифицированы по трудности. Упражнения без звездочек годятся для вводных курсов; упражнения с одной звездочкой труднее, а с двумя — еще труднее. В замечаниях по литературе в конце каждой главы делается попытка указать печатный источник для каждого алгоритма и результата, содержащихся в тексте и упражнениях.

Благодарности

Материал книги основан на набросках лекций для курсов, читавшихся авторами в Корнеллском и Принстонском университетах и в Стивенсонском технологическом институте. Авторы хотели бы поблагодарить многих людей, которые критически прочитали различные части рукописи и внесли полезные предложения. Особенно мы хотели бы поблагодарить Келлога Бута, Стэна Брауна, Стива Чена, Алена Сайфера, Арча Дейвиса, Майка Фишера, Хейнию Гаевску, Майка Гэри, Юдая Гупту, Майка Харрисона, Матта Хекта, Гарри Ханта, Дейва Джонсона, Марка Каплана, Дона Джонсона, Стива Джонсона, Брайана Кернигана, Дона Кнута, Ричарда Лэднера, Аниту Ла-Саль, Дуга Мак-Илроя, Альберта Мейера, Кристоса Пападимитриу, Билла Плогера, Джона Сэвиджа, Ховарда Зигеля, Кена Стейглица, Лэрри Стокмейера, Тома Жимански и Теодора Ена.

Мы выражаем особую благодарность Джеме Карневейл, Полине Камерон, Ханне Кресее, Эдит Персер и Руфи Сузуки за быструю и качественную перепечатку рукописи.

Авторы также благодарны фирме Bell Laboratories и университетам Корнеллскому, Принстонскому и Калифорнийскому (отделение в Беркли) за предоставленные возможности для подготовки рукописи.

Июнь 1974

Альфред Ахо
Джон Хопкрофт
Джефри Ульман

МОДЕЛИ ВЫЧИСЛЕНИЙ

Если дана задача, как найти для ее решения эффективный алгоритм? А если алгоритм найден, как сравнить его с другими алгоритмами, решающими ту же задачу? Как оценить его качество? Вопросы такого рода интересуют и программистов, и тех, кто занимается теоретическим исследованием вычислений. В этой книге мы изучим различные подходы, с помощью которых пытаются получить ответ на вопросы, подобные перечисленным.

В настоящей главе рассматриваются несколько моделей вычислительной машины — машина с произвольным доступом к памяти, машина с произвольным доступом к памяти и хранимой программой и машина Тьюринга. Эти модели сравниваются по их способности отражать сложность алгоритма, и на их основе строятся более специализированные модели вычислений, а именно: неветвящиеся арифметические программы, битовые вычисления, вычисления с двоичными векторами и деревья решений. В последнем разделе этой главы вводится язык для описания алгоритмов, называемый Упрощенным Алголом.

1.1. АЛГОРИТМЫ И ИХ СЛОЖНОСТИ

Для оценки алгоритмов существует много критериев. Чаще всего нас будет интересовать порядок роста необходимых для решения задачи времени и емкости памяти при увеличении входных данных. Нам хотелось бы связать с каждой конкретной задачей некоторое число, называемое ее *размером*, которое выражало бы меру количества входных данных. Например, размером задачи умножения матриц может быть наибольший размер матриц-сомножителей. Размером задачи о графах может быть число ребер данного графа.

Время, затрачиваемое алгоритмом, как функция размера задачи, называется *временно́й сложностью* этого алгоритма. Поведение этой сложности в пределе при увеличении размера задачи называется *асимптотической временно́й сложностью*. Аналогично можно опре-

делить *емкостную сложность* и *асимптотическую емкостную сложность*.

Именно асимптотическая сложность алгоритма определяет в итоге размер задач, которые можно решить этим алгоритмом. Если алгоритм обрабатывает входы размера n за время cn^2, где c — некоторая постоянная, то говорят, что временна́я сложность этого алгоритма есть $O(n^2)$ (читается "порядка n^2"). Точнее, говорят, что (неотрицательная) функция $g(n)$ есть $O(f(n))$, если существует такая постоянная c, что $g(n) \leqslant cf(n)$ для всех, кроме конечного (возможно, пустого) множества, неотрицательных значений n.

Можно было бы подумать, что колоссальный рост скорости вычислений, вызванный появлением нынешнего поколения цифровых вычислительных машин, уменьшит значение эффективных алгоритмов. Однако происходит в точности противоположное. Так как вычислительные машины работают все быстрее и мы можем решать все бóльшие задачи, именно сложность алгоритма определяет то увеличение размера задачи, которое можно достичь с увеличением скорости машины.

Допустим, у нас есть пять алгоритмов A_1—A_5 со следующими временными сложностями:

Алгоритм	Временная сложность
A_1	n
A_2	$n \log n$[1])
A_3	n^2
A_4	n^3
A_5	2^n

Здесь временная сложность — это число единиц времени, требуемого для обработки входа размера n. Пусть единицей времени будет одна миллисекунда. Тогда алгоритм A_1 может обработать за одну секунду вход размера 1000, в то время как A_5 — вход размера не более 9. На рис. 1.1 приведены размеры задач, которые можно решить за одну секунду, одну минуту и один час каждым из этих пяти алгоритмов.

Предположим, что следующее поколение вычислительных машин будет в 10 раз быстрее нынешнего. На рис. 1.2 показано, как возрастут размеры задач, которые мы сможем решить благодаря этому увеличению скорости. Заметим, что для алгоритма A_5 десятикратное увеличение скорости увеличивает размер задачи, которую можно решить, только на три, тогда как для алгоритма A_3 размер задачи более чем утраивается.

Вместо эффекта увеличения скорости рассмотрим теперь эффект применения более действенного алгоритма. Вернемся к рис. 1.1.

[1]) Если не оговорено противное, все логарифмы в этой книге берутся по основанию 2.

Алгоритм	Временна́я сложность	Максимальный размер задачи		
		1 с	1 мин	1 ч
A_1	n	1000	6×10^4	$3{,}6\times10^6$
A_2	$n\log n$	140	4893	$2{,}0\times10^5$
A_3	n^2	31	244	1897
A_4	n^3	10	39	153
A_5	2^n	9	15	21

Рис. 1.1. Границы размеров задач, определяемые скоростью роста сложности.

Если в качестве основы для сравнения взять 1 мин, то, заменяя алгоритм A_4 алгоритмом A_3, можно решить задачу, в 6 раз большую, а заменяя A_4 на A_2, можно решить задачу, большую в 125 раз. Эти результаты производят гораздо большее впечатление, чем двукратное улучшение, достигаемое за счет десятикратного увеличения скорости. Если в качестве основы для сравнения взять 1 ч, то различие оказывается еще значительнее. Отсюда мы заключаем, что асимптотическая сложность алгоритма служит важной мерой качественности алгоритма, причем такой мерой, которая обещает стать еще важнее при последующем увеличении скорости вычислений.

Несмотря на то что основное внимание здесь уделяется порядку роста величин, надо понимать, что большой порядок роста сложности алгоритма может иметь меньшую мультипликативную посто-

Алгоритм	Временна́я сложность	Максимальный размер задачи	
		до ускорения	после ускорения
A_1	n	s_1	$10s_1$
A_2	$n\log n$	s_2	Примерно $10s_2$ для больших s_2
A_3	n^2	s_3	$3{,}16s_3$
A_4	n^3	s_4	$2{,}15s_4$
A_5	2^n	s_5	$s_5+3{,}3$

Рис. 1.2. Эффект десятикратного ускорения.

янную, чем малый порядок роста сложности другого алгоритма. В таком случае алгоритм с быстро растущей сложностью может оказаться предпочтительнее для задач с малым размером — возможно, даже для всех задач, которые нас интересуют. Например, предположим, что временные сложности алгоритмов A_1, A_2, A_3, A_4 и A_5 в действительности равны соответственно $1000n$, $100n \log n$, $10n^2$, n^3 и 2^n. Тогда A_5 будет наилучшим для задач размера $2 \leqslant n \leqslant 9$, A_3— для задач размера $10 \leqslant n \leqslant 58$, A_2— при $59 \leqslant n \leqslant 1024$, а A_1— при $n > 1024$.

Прежде чем продолжать обсуждение алгоритмов и их сложностей, мы должны точно определить модель вычислительного устройства, используемого для реализации алгоритмов, а также условиться, что понимать под элементарным шагом вычисления. К сожалению, нет такой вычислительной модели, которая была бы хороша во всех ситуациях. Одна из основных трудностей связана с размером машинных слов. Например, если допустить, что машинное слово может быть целым числом произвольной длины, то всю задачу можно закодировать одним числом в виде одного машинного слова. Если же допустить, что машинное слово имеет фиксированную длину, то возникают трудности даже просто запоминания произвольно больших чисел, не говоря уже о других проблемах, которые часто удается обойти, если решаются задачи скромного размера. Для каждой задачи надо выбирать подходящую модель, которая точно отразит действительное время вычисления на реальной вычислительной машине.

В следующих разделах мы обсудим несколько основных моделей вычислительных устройств, наиболее важными из которых являются машина с произвольным доступом к памяти, машина с произвольным доступом к памяти и хранимой программой и машина Тьюринга. Эти три модели эквивалентны с точки зрения принципиальной вычислимости, но не с точки зрения скорости вычислений.

Быть может, наиболее важным мотивом, побудившим рассмотрение формальных моделей вычисления, было желание раскрыть подлинную вычислительную сложность различных задач. Нам хотелось бы получить нижние оценки на время вычислений. Чтобы показать, что не существует алгоритма, выполняющего данное задание менее чем за определенное время, необходимо точное и подчас высоко специализированное определение того, что есть алгоритм. Одним из примеров такого определения служат машины Тьюринга в разд. 1.6.

Для описания алгоритмов желательно иметь обозначения, более естественные и легче понимаемые, чем программа для машины с произвольным доступом к памяти, машины с произвольным доступом к памяти и хранимой программой или машины Тьюринга. Поэтому мы введем также язык высокого уровня, называемый Упрощенным Алголом. Именно этот язык будет использоваться во всей книге для

описания алгоритмов. Однако, чтобы понимать вычислительную сложность алгоритма, написанного на Упрощенном Алголе, мы должны соотнести Упрощенный Алгол с более формальными моделями. Это будет сделано в последнем разделе настоящей главы.

1.2. МАШИНЫ С ПРОИЗВОЛЬНЫМ ДОСТУПОМ К ПАМЯТИ

Машина с произвольным доступом к памяти (или, иначе, равнодоступная адресная машина — сокращенно РАМ) моделирует вычислительную машину с одним сумматором, в которой команды программы не могут изменять сами себя.

РАМ состоит из входной ленты, с которой она может только считывать, выходной ленты, на которую она может только записывать, и памяти (рис. 1.3). Входная лента представляет собой последовательность клеток, каждая из которых может содержать целое число (возможно, отрицательное). Всякий раз, когда символ считывается с входной ленты, ее читающая головка сдвигается на одну клетку вправо. Выход представляет собой ленту, на которую машина может только записывать; она разбита на клетки, которые вначале все пусты. При выполнении команды записи в той клетке выходной ленты, которую в текущий момент обозревает ее головка, печатается целое число и головка затем сдвигается на одну клетку вправо. Как только выходной символ записан, его уже нельзя изменить.

Память состоит из последовательности регистров r_0, r_1, . . ., r_i, . . ., каждый из которых способен хранить произвольное целое

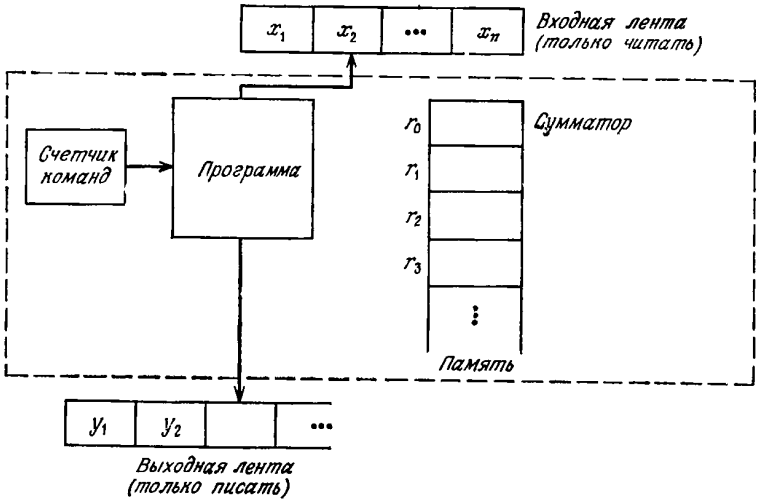

Рис. 1.3. Машина с произвольным доступом к памяти.

число. На число регистров, которые можно использовать, мы не устанавливаем верхней границы. Такая идеализация допустима в случаях, когда

1) размер задачи достаточно мал, чтобы она поместилась в основную память вычислительной машины,
2) целые числа, участвующие в вычислении, достаточно малы, чтобы их можно было помещать в одну ячейку.

Программа для РАМ (или РАМ-программа) не записывается в память. Поэтому мы предполагаем, что программа не изменяет сама себя. Программа является, в сущности, последовательностью (возможно) помеченных команд. Точный тип команд, допустимых в программе, не слишком важен, пока они напоминают те, которые обычно встречаются в реальных вычислительных машинах. Мы предполагаем, что имеются арифметические команды, команды ввода-вывода, косвенная адресация (например, для индексации массивов) и команды разветвления. Все вычисления производятся в первом регистре r_0, называемом *сумматором*, который, как и всякий другой регистр памяти, может хранить произвольное целое число. Пример набора команд для РАМ показан на рис. 1.4. Каждая команда состоит из двух частей — *кода операции* и *адреса*.

В принципе можно было бы добавить к нашему набору любые другие команды, встречающиеся в реальных вычислительных машинах, такие, как логические или литерные операции, и при этом порядок сложности задач не изменится. Читателю разрешается думать, что набор команд дополнен так, как это его устраивает.

Операнд может быть одного из следующих типов:

1) $=i$ означает само целое число i и называется литералом,
2) i — содержимое регистра i (i должно быть неотрицательным),
3) $*i$ означает косвенную адресацию, т. е. значением операнда служит содержимое регистра j, где j — целое число, находящееся в регистре i; если $j<0$, машина останавливается.

Эти команды хорошо знакомы всякому, кто программировал на языке ассемблера. Можно определить значение программы P с помощью двух объектов: отображения c из множества неотрицательных целых чисел в множество целых чисел и «счетчика команд», который определяет очередную выполняемую команду. Функция c есть *отображение памяти*, а именно $c(i)$ — целое число, содержащееся в регистре i (*содержимое* регистра i).

Вначале $c(i)=0$ для всех $i\geqslant 0$, счетчик команд установлен на первую команду в P, а выходная лента пуста. После выполнения k-й команды из P счетчик команд автоматически переходит на $k+1$ (т. е. на очередную команду), если k-я команда не была командой вида JUMP, HALT, JGTZ или JZERO.