ПЕДАГОГИЧЕСКОЕ ОБРАЗОВАНИЕ

Д. Ш. Матрос
Г. Б. Поднебесова

ТЕОРИЯ АЛГОРИТМОВ

БИНОМ

ПЕДАГОГИЧЕСКОЕ ОБРАЗОВАНИЕ

Д. Ш. Матрос
Г. Б. Поднебесова

ТЕОРИЯ АЛГОРИТМОВ

Учебное пособие

Москва
БИНОМ. Лаборатория знаний
2 0 0 7

УДК 570
ББК 22.12
М33

Матрос Д. Ш.

М33 Теория алгоритмов: Учебное пособие / Д. Ш. Матрос, Г. Б. Поднебесова — М.: БИНОМ. Лаборатория знаний, 2007. — 196 с.: ил.
ISBN 978-5-94774-226-8

Учебное пособие по курсу «Теория алгоритмов» для педагогических вузов по специальности «Информатика», полностью соответствующее стандарту.

Изложение имеет четкую логическую структуру и охватывает следующие темы: понятие алгоритма, машина Тьюринга, примитивно-рекурсивные функции, нормальные алгоритмы, вычислимость и разрешимость, сложность вычислений, NP-полные задачи. Каждая тема сопровождается тестовыми заданиями и упражнениями.

Для студентов и преподавателей педагогических вузов, учителей общеобразовательных школ.

УДК 570
ББК 22.12

По вопросам приобретения обращаться:

«БИНОМ. Лаборатория знаний» (495) 157-52-72, e-mail: Lbz@aha.ru

http://www.Lbz.ru

ISBN 978-5-94774-226-8

Оглавление

Предисловие

В соответствии со стандартами высшего профессионального образования второго поколения по специальности 030100 (Информатика в педагогических университетах и институтах) введен новый предмет «Теория алгоритмов».

Данный учебник содержит материал, полностью соответствующий требованиям государственного образовательного стандарта по этому предмету и реализующий основные цели:

- познакомить читателя с максимально широким кругом понятий теории алгоритмов. Количество определяемых понятий и специальных терминов превышает количество понятий, обсуждаемых более детально. Тем самым у студентов формируется терминологический запас, необходимый для самостоятельного изучения специальной математической программистской литературы;
- сообщить читателю необходимые конкретные сведения из теории алгоритмов, представляемые стандартом. Разбор доказательств приведенных утверждений, ответы на тестовые задания и выполнение упражнений позволяют студенту овладеть методами, наиболее употребляемыми при решении практических задач, и понять основные идеи современной теории алгоритмов;
- получить запас алгоритмически неразрешимых задач (проблем). Изучение таких задач необходимо программисту для уменьшения трудозатрат на «изобретение велосипеда» и обогащения его навыками конструирования алгоритмов.

Реализация этих целей позволит будущему учителю информатики понять важность основных положений теории алгоритмов для современной практики, осознать, что без их

глубокого знания невозможно построение современного программного обеспечения, необходимого для научных и прикладных исследований.

Сформулированные цели связаны также и с тем, что, к сожалению, даже многие учителя информатики и программисты не могут объяснить, например, почему за последние пятьдесят лет практически ничего не изменилось в процессе отладки программ, что особенно контрастирует с колоссальным прогрессом в области собственно программирования.

В последнее время теория алгоритмов — активно развивающаяся область, лежащая на стыке математики и информатики. Полученные результаты нашли отражение в ряде монографий. В первую очередь это вопросы, связанные со сложностью алгоритмов, и теория так называемых NP-полных задач. Но учебной литературы для педагогических вузов, которая освещала бы все эти проблемы (от классических построений до результатов последних лет), нет.

Все это привело к следующей структуре книги.

В первой главе вводится неформальное понятие алгоритма, формулируются основные требования к нему, напоминаются основные определения из других разделов математики. Рассматривается алгоритм как программа для компьютера и структурная теорема Бома–Джакопини.

Следующие четыре главы посвящены наиболее принципиальным подходам при введении понятия алгоритма (машины Тьюринга, рекурсивные функции, нормальные алгоритмы Маркова, машины с неограниченными регистрами (МНР)). Теоретическая строгость изложения везде доводится до интерпретации полученных результатов для практики современного программирования. Более того, показано, что с помощью любой из предложенных теорий можно «запрограммировать» любую реализованную на компьютере задачу, а вот доказать несуществование компьютерной программы для решения задачи можно только с помощью какой-то из этих теорий. Такой методологический подход очень важен для будущих учителей.

В шестой главе показывается эквивалентность различных теорий алгоритмов и доказывается алгоритмическая неразрешимость некоторых проблем в теории вычислимости. При этом подчеркивается, что основная причина — общность в постановке тех или иных задач. То есть при определенной конкретизации задачи становятся вполне решаемыми, о чем и свидетельствует опыт программирования на компьютере.

Седьмая глава посвящена теоретическим основам и приложениям к языкам программирования рекурсивных операторов.

В восьмой главе рассматривается вопрос вычислимости функции за какое-то определенное время. Знаменитая теорема Блюма об ускорении показывает, что «естественный» путь определения сложности функции через сложность реализующей ее программы невозможен. В связи с этим вводятся специальные классы сложности и рассматривается класс хорошо знакомых нам со школы элементарных функций.

В заключительной главе дается введение в теорию NP-полных задач. Приводится большое количество примеров и формулируется одна из важнейших проблем современной математики — как соотносятся между собой классы P и NP.

Каждая глава заканчивается контрольным тестом и упражнениями. Как показывает опыт, выполнение упражнений позволяет лучше усвоить пройденный материал, а тестовые задания дают возможность организовать рейтинговую систему в процессе обучения предмету. В книге конец упражнения обозначен символом $\triangle$, а знак $\square$ указывает на завершение доказательства теоремы.

При изложении авторы следовали монографиям С. К. Клини, О. П. Кузнецова и Г. М. Адельсон-Вельского, Н. Катленда, А. И. Мальцева, А. А. Маркова и Н. И. Нагорного, М. Гэри и Д. Джонсона и др. Этими работами мы и рекомендуем пользоваться для дальнейшего изучения материала.

Данная книга предназначена для будущих учителей информатики, которые уже прошли такие предметы, как высшая математика, математическая логика, дискретная математика, теория вероятностей, программное обеспечение, программирование. Поэтому предполагается, что читатель не испытает затруднений в понимании математических текстов.

Авторы будут рады любым замечаниям и предложениям, которые можно направлять по адресу:

454080, г.Челябинск, пр. Ленина, 69,

Челябинский государственный педагогический университет, факультет информатики.

E-mail: matros@cspu.ru

Глава I

Предварительные обсуждения

Данная глава посвящена изложению предварительных результатов, которые на неформальном уровне вводят нас в круг необходимых проблем.

В первом пункте дается интуитивное понятие алгоритма, излагаются основные требования к нему. В дальнейшем, при различных уточнениях этого понятия, мы будем показывать, что все предлагаемые конструкции удовлетворяют сформулированным требованиям. Подробно рассматриваются блок-схемы как один из самых распространенных и наглядных способов представления алгоритмов. Описываются основные идеи различных подходов к уточнению понятия алгоритма, которые будут рассмотрены в последующих главах.

Во втором пункте вводятся, скорее напоминаются, основные определения из других математических дисциплин, которые будут использоваться в дальнейшем.

В третьем пункте алгоритм рассматривается как программа для компьютера. Из структурной теоремы следует, что необходимо показать возможность реализации каждой из трех базовых структур при любом уточнении понятия алгоритма. Отсюда будет следовать возможность реализации любой программы для компьютера средствами модели, уточняющей понятие алгоритма.

1.1. Неформальное понятие алгоритма

С *алгоритмами*, т. е. эффективными процедурами [8], однозначно приводящими к результату, математика имела дело всегда. Школьные методы умножения «столбиком» и деления «углом», метод исключения неизвестных при решении системы линейных уравнений, правило дифференцирования сложных функций, способ построения треугольника по трем заданным сторонам — все это алгоритмы. Однако

пока математика имела дело в основном с числами и вычислениями и понятие алгоритма отождествлялось с понятием метода вычисления, потребности в изучении самого этого понятия не возникало. Традиции организации вычислений складывались веками и стали составной частью общей научной культуры в той же степени, что и элементарные навыки логического мышления. Все многообразие вычислений комбинировалось из 10–15 четко определенных операций арифметики, тригонометрии и анализа. Поэтому понятие метода вычисления считалось изначально ясным и не нуждалось в специальных исследованиях.

До середины XIX в. единственной областью математики, работавшей с нечисловыми объектами, была геометрия, и как раз она, не имея возможности опираться на вычислительную интуицию человека, резко отличалась от остальной математики повышенными требованиями к строгости своих рассуждений. До сих пор любой современный семиклассник, для которого математика — это мир вычислений, мучительно привыкает к понятиям доказательства и математического построения и никак не может понять, зачем доказывать равенство отрезков, когда проще измерить, и зачем строить перпендикуляр с помощью циркуля и линейки, когда есть угольник с «готовым» прямым углом или транспортир.

Такое же мучительное привыкание к новым, более жестким требованиям строгости началось в математике во второй половине XIX века. Оно стимулировалось в основном математикой нечисловых объектов — открытием неэвклидовой геометрии, появлением абстрактных алгебраических теорий типа теорий групп и т. д. Одним из решающих обстоятельств, приведших к пересмотру оснований математики, т. е. принципов, лежащих в основе математических рассуждений, явилось создание Г. Кантором теории множеств. Довольно быстро стало ясно, что понятия теории множеств в силу своей общности лежат в основе всего здания математики. Однако почти столь же быстро было показано, что некоторые кажущиеся вполне естественные рассуждения в рамках этой теории приводят к неразрешимым противоречиям — парадоксам теории множеств (более подробно см. [6]). Все это потребовало точного изучения принципов математических рассуждений (до сих пор казавшихся интуитивно ясными) математическими же свойствами. Возникла особая отрасль математики — основание математики, или метаматематика.

Опыт парадоксов теории множеств научил математику крайне осторожно обращаться с бесконечностью и по возможности даже о бесконечности рассуждать с помощью финитных методов. Сущность финитного подхода заключается в том, что он допускает только конечный набор действий над конечным числом объектов. Выявление того, какие объекты и действия над ними следует считать точно определенными, какими свойствами и возможностями обладают комбинации элементарных действий, что можно и чего нельзя сделать с их помощью — все это стало предметом теории алгоритмов и формальных систем, которая первоначально возникла в рамках математики и стала важнейшей ее частью. Главным внутриматематическим приложением теории алгоритмов явилось доказательство невозможности алгоритмического (т. е. точного и однозначного) решения некоторых математических проблем. Такие доказательства (да и точные формулировки доказываемых утверждений) неосуществимы без точного понятия алгоритма.

Пока техника использовала вычислительные методы, эти высокие проблемы чистой математики ее мало интересовали. В технику термин «алгоритм» пришел вместе с кибернетикой. Если понятие метода вычисления не нуждалось в пояснениях, то понятие процесса управления пришлось вырабатывать практически заново. Понадобилось осознавать, каким требованиям должна удовлетворять последовательность действий (или ее описание), чтобы считаться конструктивно заданной, т. е. иметь право называться алгоритмом. В этом осознании огромную помощь инженерной интуиции оказала практика использования вычислительных машин, сделавшая понятие алгоритма ощутимой реальностью. С точки зрения современной практики *алгоритм — это программа*, а критерием алгоритмичности процесса является возможность его запрограммировать. Именно благодаря этой реальности алгоритма, а также потому, что подход инженера к математическим методам всегда был конструктивным, понятие алгоритма в технике за короткий срок стало необычайно популярным (быть может, даже больше, чем в самой математике).

Однако у всякой популярности есть свои издержки, в повседневной практике слово «алгоритм» употребляется слишком широко, теряя зачастую свой точный смысл. Приблизительные описания понятия «алгоритм» (вроде того, которое приведено в первой фразе этого пункта) часто принимаются за точные определения. В результате за алго-

ритм зачастую выдается любая инструкция, разбитая на шаги. Появляются такие дикие словосочетания, как «алгоритм изобретения» (а ведь наличие «алгоритма изобретения» означало бы конец изобретательства как творческой деятельности).

Ясное представление о том, что такое алгоритм, важно, конечно, не только для правильного словоупотребления. Оно важно и при разработке конкретных алгоритмов, особенно когда имеется в виду их последующее программирование. Чтобы ориентироваться в море алгоритмов, необходимо уметь сравнивать различные алгоритмы решения одних и тех же задач, причем не только по качеству решений, но и по характеристикам самих алгоритмов (числу действий, расходу памяти и т. д.). Такое сравнение невозможно без введения точного языка для обсуждения всех этих вопросов; иначе говоря, сами алгоритмы должны стать такими же предметами точного исследования, как и те объекты, для работы с которыми они предназначены.

Строгое исследование основных понятий теории алгоритмов начнется в следующей главе. Прежде чем приступить к нему, обсудим на неформальном уровне некоторые основные принципы, по которым строятся алгоритмы, и выясним, что же именно в понятии алгоритма нуждается в уточнении.

1.1.1. Основные требования к алгоритмам

Отметим несколько общих черт алгоритмов, ясно вырисовывающихся из вышесказанного и часто признающихся характерными [7,11] для понятия алгоритма:

a) алгоритм — это процесс последовательного построения величин, идущий в дискретном времени таким образом, что в начальный момент задается исходная конечная система величин, а в каждый следующий момент система величин получается по определенному закону (программе) из системы величин, имевшихся в предыдущий момент времени (*дискретность алгоритма*);

b) система величин, получаемых в какой-то (не начальный) момент времени, однозначно определяется системой величин, полученных в предшествующие моменты времени (*детерминированность алгоритма*);

c) закон получения последующей системы величин из предшествующей должен быть простым и локальным (*элементарность шагов алгоритма*), типичный пример множества элементарных действий — система команд ЭВМ;
d) если способ получения последующей величины из какой-нибудь заданной величины не дает результата, то должно быть указано, что надо считать результатом алгоритма (*результативность алгоритма*);
e) начальная система величин может выбираться из некоторого потенциально бесконечного множества (*массовость алгоритма*).

Понятие алгоритма, в какой-то мере определяемое этими требованиями, конечно, нестрогое: в формулировках требований встречаются слова «способ», «величина», «простой», «локальный», точный смысл которых не установлен. В дальнейшем это нестрогое понятие алгоритма будет называться **непосредственным** или **интуитивным понятием алгоритма**.

Сделаем несколько комментариев к описанным требованиям. Любой алгоритм применяется к исходным данным и выдает результат. В привычных технических терминах это означает, что алгоритм имеет входы и выходы. Кроме того, в ходе работы алгоритма появляются промежуточные результаты, которые используются в дальнейшем. Таким образом, каждый алгоритм имеет дело с данными — входными, промежуточными и выходными. Поскольку мы собираемся уточнять понятие алгоритма, нужно уточнить и понятие данных, т. е. указать, каким требованиям должны удовлетворять объекты, чтобы алгоритмы могли с ними работать.

Ясно, что эти объекты должны быть четко определены и отличимы как друг от друга, так и от «необъектов». Во многих случаях хорошо понятно, что это значит: к таким алгоритмическим объектам относятся числа, векторы, матрицы смежностей графов, формулы. Изображения (например, рисунок графа) представляются менее естественными в качестве алгоритмических объектов. Если говорить о графе, то дело даже не в том, что в рисунке больше несущественных деталей и два человека один и тот же граф изобразят по-разному (в конце концов, разные матрицы смежности тоже могут задавать один и тот же граф с точностью до изоморфизма), а в том, что матрица смежности легко разбивается на элементы, причем из элементов всего двух видов (нулей и

единиц) строят матрицы любых графов, тогда как разбить на элементы рисунок гораздо труднее. Наконец, с такими объектами, как хорошая книга или осмысленное утверждение, с которыми легко управляется любой человек (но каждый по-своему!), алгоритм работать не сможет, пока они не будут описаны как данные с помощью других, более подходящих объектов.

Вместо того чтобы пытаться дать общее словесное определение четкой определенности объекта, в теории алгоритмов фиксируют конкретные конечные наборы исходных объектов (называемых элементарными) и конечный набор средств построения других объектов из элементарных. Набор элементарных объектов образует конечный *алфавит* исходных символов (цифр, букв и т. д.), из которых строятся другие объекты.

Типичным средством построения являются индуктивные определения, указывающие, как строить новые объекты из уже построенных. Простейшее индуктивное определение — это определение некоторого множества слов, классическим примером которого служит определение идентификатора в ПАСКАЛе: идентификатор — это либо буква, либо идентификатор, к которому приписана справа буква или цифра. Слова конечной длины в конечных алфавитах (в частности, числа) — самый простой тип алгоритмических данных, а число символов в слове (длина слова) — естественная единица измерения объема обрабатываемой информации. Более сложный случай алгоритмических объектов — формулы. Они также определяются индуктивно и также являются словами в конечном алфавите, однако не каждое слово в этом алфавите является формулой. В этом случае обычно основным алгоритмам предшествуют вспомогательные, которые проверяют, удовлетворяют ли исходные данные нужным требованиям. Такая проверка называется синтаксическим анализом.

Данные для своего размещения требуют *памяти*. Память обычно считается однородной и дискретной, т. е. состоит из одинаковых ячеек, причем каждая ячейка может содержать один символ алфавита данных. Таким образом, единицы измерения объема данных и памяти согласованы. При этом память может быть бесконечной. Вопрос о том, нужна ли отдельная память для каждого из трех видов данных (входных, выходных и промежуточных), решается по-разному.

Требование результативности, в частности, предполагает, что всякий, кто предъявляет алгоритм решения некоторой задачи, например вычисления функции $f(x)$, обязан показать, что алгоритм останавливается после конечного числа шагов (как говорят, сходится) для любого x из области задания f. Однако проверить результативность (сходимость) гораздо труднее, чем требования, изложенные в других пунктах. В отличие от них сходимость обычно не удается установить простым просмотром описания алгоритма; общего же метода проверки сходимости, пригодного для любого алгоритма A и любых входных данных x, как будет показано далее, вообще не существует.

Если алгоритм используется для вычисления значений числовой функции, то эта функция называется *эффективно вычислимой*, или *алгоритмически вычислимой*, или просто *вычислимой*.

Следует различать:

1) описание алгоритма (инструкцию или программу);
2) механизм реализации алгоритма (например, компьютер), включающий средства пуска, остановки, реализации элементарных шагов, выдачи результатов и обеспечения детерминированности, т. е. управления ходом вычисления;
3) процесс реализации алгоритма, т. е. последовательность шагов, которая будет порождена при применении алгоритма к конкретным данным.

Будем предполагать, что описание алгоритма и механизм его реализации конечны (память, как уже говорилось, может быть бесконечной, но она не включается в механизм). Требование конечности процесса реализации совпадает с требованием результативности.

Пример 1. Рассмотрим следующую задачу: дана последовательность P из n положительных чисел (n — конечное, но произвольное число); требуется упорядочить их, т. е. построить последовательность R, в которой эти же числа расположены в порядке возрастания. Почти сразу же приходит в голову следующий простой способ ее решения: просматриваем P и находим в ней наименьшее число; вычеркиваем его из P и выписываем его в качестве первого числа R; снова обращаемся к P и находим в ней наименьшее число; приписываем его справа к полученной части R и так далее, до тех пор, пока из P не будут вычеркнуты все числа.

Возникает естественный вопрос: что значит «и так далее»? Для большей ясности перепишем описание способа решения в более четкой форме, разбив его на шаги и указав переходы между шагами.

Шаг 1. Ищем в P наименьшее число.

Шаг 2. Найденное число приписываем справа к R (в начальный момент R пуста) и вычеркиваем его из P.

Шаг 3. Если в P нет чисел, то переходим к шагу 4. В противном случае переходим к шагу 1.

Шаг 4. Конец. Результатом считать последовательность R, построенную к данному моменту. $\triangle$

Большинство читателей сочтет такое описание достаточно ясным (и даже излишне формальным), чтобы, пользуясь им, однозначно получить нужный результат. Однако это впечатление ясности опирается на некоторые неявные предположения, к правильности которых мы привыкли, но которые нетрудно нарушить. Например, что значит «дана последовательность чисел»? Является ли таковой последовательность $\sqrt[7]{3}$, $\sqrt[5]{2}$, $(1,2)^{\pi}$? Очевидно, да, однако в нашем описании ничего не сказано, как найти наименьшее число среди таких чисел. В нем вообще не говорится о том, как искать наименьшие числа, по-видимому, предполагается, что речь идет о числах, представленных в виде десятичных дробей, и что известно, как их сравнивать.

Итак, необходимо уточнить формы представления данных. При этом нельзя просто заявить, что допустимо любое представление чисел. Ведь для каждого представления существует свой алфавит (который, помимо цифр, может включать запятые, скобки, знаки операций и функций и т. д.) и свой способ сравнения чисел (например, перевод в десятичную дробь), тогда как конечность алфавита требует фиксировать его заранее, а конечность описания алгоритма позволяет включить в него лишь заранее фиксированное число способов сравнения. Фиксация представления чисел в виде десятичных дробей также не решает всех проблем. Сравнение 10–20-разрядных чисел уже не может считаться элементарным действием: сразу нельзя сказать, какое из чисел больше: 90811557001,15 или 32899901467,0048. В машинных алгоритмах само представление числа еще требует дальнейшего уточнения: нужно, во-первых, ограничить число разрядов (цифр) в числе, так как от этого зависит, сколько ячеек памяти будет занимать число, а во-вторых, договориться о способе размещения десятичной запятой в числе,

т. е. выбрать представление в виде числа с фиксированной или плавающей запятой, поскольку способы обработки этих двух представлений различны. Наконец, любой, кто имел дело с программированием, отметит, что на шаге 1 требуется узнать две вещи: само наименьшее число (чтобы записать его в R) и его место в P, т. е. его адрес в той части памяти, где хранится P (чтобы вычеркнуть его из P), а следовательно, нужно иметь средства указания этого адреса.

Таким образом, даже в этом простом примере несложный анализ показывает, что описанию, которое выглядит вполне ясным, еще далеко до алгоритма. Мы столкнулись здесь с необходимостью уточнить почти все основные характеристики алгоритма, которые отмечались ранее: алфавит данных и форму их представления, память и размещение в ней элементов P и R, элементарные шаги (поскольку шаг 1 явно не элементарен). Кроме того, становится ясным, что выбор механизма реализации (скажем, человека или компьютера) будет влиять и на сам характер уточнения: у человека требования к памяти, представлению данных и к элементарности шагов гораздо более слабые и «укрупненные», отдельные незначительные детали он может уточнить сам.

Пожалуй, только два требования к алгоритмам в приведенном описании выполнены в достаточной мере (они-то и создают впечатление ясности). Довольно очевидна сходимость алгоритма: после выполнения шагов 1 и 2 либо работа заканчивается, либо из P вычеркивается одно число; поэтому ровно после n выполнений шагов 1 и 2 из P будут вычеркнуты все числа и алгоритм остановится, а R будет результатом. Кроме того, не вызывает сомнения детерминированность: после каждого шага ясно, что делать дальше, если учесть, что здесь и в дальнейшем используется общепринятое соглашение — если шаг не содержит указаний о дальнейшем переходе, то после него выполняем шаг, следующий за ним в описании. Поскольку использованные в примере средства обеспечения детерминированности носят довольно общий характер, остановимся на них несколько подробнее.

1.1.2. Блок-схемы алгоритмов

Связи между шагами можно изобразить в виде графа. Для примера из предыдущего пункта граф изображен на рис. 1.1.