Московский государственный технический университет имени Н.Э. Баумана

Факультет «Фундаментальные науки»
Кафедра «Математическое моделирование»

А.Н. Канатников

МАТЕМАТИЧЕСКАЯ ЛОГИКА И ТЕОРИЯ АЛГОРИТМОВ

Конспект лекций

Для студентов кафедры ИУ9

Москва
2009

8. ОСНОВНЫЕ ПОНЯТИЯ ТЕОРИИ АЛГОРИТМОВ

8.1. Основные характеристики алгоритма

Интуитивное понятие алгоритма. Его отличительные черты: дискретность, детерминированность, элементарность шагов, направленность, массовость). Работа алгоритма как вычисление функции. Понятие вычислимой функции. Алгоритм как функционирование абстрактной машины.

Алгоритм — одно из фундаментальных понятий не только современной математики, но и всего естествознания. Развитие информационных технологий и группы наук, обеспечивающих эти технологии, привело к тому, что понятие „алгоритм“ стало одним из ключевых.

Фундаментальность этого понятия означает, что мы не можем рассчитывать на точное определение алгоритма: это означало бы, что есть более фундаментальные понятия. Но эта ситуация с понятием „алгоритм“ далеко не уникальна. Скажем, как определить, что такое энергия? Да, в теоретической механике, объектом исследований которой являются относительно простые физические объекты, можно точно сформулировать что такое энергия. В других физических дисциплинах это понятие приходится уточнять и расширять. В конечном счете оказывается, что энергия — нечто большее, нежели физический термин, это некоторая сторона нашего мироощущения.

То же самое можно сказать о понятии „алгоритм“. Мы можем выделить лишь некоторые отличающее это понятие черты. Какие же?

Можно сказать, что алгоритм — это точное предписание, задающее вычислительный процесс, который начинается с произвольного объекта некоторой совокупности и направлен на получение полностью определяемого исходным объектом результата.

Приведенное описание понятия можно уточнить, указав некоторые характерные черты алгоритма.

1. Дискретность: алгоритмический процесс расчленяется на отдельные шаги ограниченной сложности, каждый из которых состоит в непосредственной переработке полученных к этому моменту результатов.

2. Детерминированность: непосредственная обработка результатов на каждом шаге полностью определяется исходным объектом и результатами, полученными на предыдущих шагах*.

3. Направленность: алгоритмический процесс, примененный к исходному объекту, дает „решение задачи“ — заключительный объект. По-другому: на каждом шаге алгоритмического процесса известно, что является результатом.

4. Массовость: исходный объект может выбираться произвольно из некоторой бесконечной совокупности.

Описание алгоритма даже с учетом перечисления характерных черт очень расплывчатое. Впрочем это не мешает, когда речь идет о поиске решения задачи. Конкретное описание решения можно непосредственно проанализировать и сказать, является это описание алгоритмом или нет.

Приведем некоторые примеры.

*К этому иногда добавляют условие локальности, заключающееся в том, что на каждом шаге используются результаты только ограниченного количества предыдущих шагов. Однако в силу конечности всех рассматриваемых процессов условие конечности всегда можно считать выполненным. Это условие носит скорее методологический характер.

Пример 8.1. Самые первые примеры алгоритмов — алгоритмы выполнения арифметических операций над числами, записанными в десятичной (или какой-то иной) системе исчисления. Попутно отметим, что алгоритмы эти работают не с самими числами,а с их представлениями. Можно себе представить, как надо выполнять арифметические операции над числами, записанными римскими цифрами. Во всяком случае ясно, что это будут совсем другие алгоритмы, заметно отличающиеся от привычных нам.

Пример 8.2. Алгоритм Евклида вычисления наибольшего общего делителя — один из самых древних в истории математики. Чтобы найти наибольший общий делитель натуральных чисел p и q, делим p на q с остатком: $p = \alpha_1 q + \beta_1$. Затем второе число q делим первый остаток β_1 с остатком: $q = \alpha_2 beta_1 + \beta_2$. Затем делим β_1 на β_2 и т.д. Наступит момент, когда очередной остаток равен нулю: $\beta_{k-1} = \alpha_{k+1}\beta_k$. В этот момент процедура прекращается, а пользователю предъявляется решение: наибольший общий делитель — это число β_k.

Весь этот процесс можно представить себе как работу некоей машины-делителя. В начале мы вносим в нее исходные числа p и q и запускаем процесс. После остановки процесса извлекаем результат β_k. Машина будет работать при любых натуральных числах и всегда будет давать ответ.

Пример 8.3. Еще один известный алгоритм — алгоритм Гаусса решения линейных систем. Ограничимся крамеровскими системами, хотя это и не принципиально. Отметим, что этот алгоритм в некоторых ситуациях может приводит к делению на нуль и тем самым не давать решения, хотя решение существует (вопрос, для каких систем алгоритм Гаусса не работает?). Модификация алгоритма Гаусса с выбором главного элемента дает решение для любой крамеровской системы. Впрочем, последнее утверждение тоже следует считать условным: при работе с числами, имеющими фиксированное число знаков, алгоритм может аварийно останавливаться в двух случаях: переполнение и деление на нуль.

Вообще говоря, в качестве объектов, которые перерабатываются алгоритмом, могут выступать в принципе любые объекты. Однако отметим два обстоятельства.

Во-первых, все объекты, которые изучает математика могут быть представлены в числовой форме (в виде чисел и совокупностей чисел), причем все в конечном счете сводится к натуральным числам. Так, функция действительного переменного — это некоторое подмножество в $\mathbb{R}^2$ (то, что мы называем графиком функции), т.е. некоторое множество пар действительных чисел. В свою очередь, каждое действительное число можно представить в виде бесконечной десятичной дроби. Метод координат позволяет все геометрические объекты перевести в числовую форму.

Во-вторых, используемые на практике алгоритмы работают не с самими объектами, а с их представлениями в том или ином языке. Все наши процессы обработки сводятся к переработке групп чисел с фиксированным числом разрядов (целых чисел), записанных в той или иной системе исчисления. Поэтому алгоритм можно интерпретировать как функцию нескольких переменных, аргументы и значения которой являются натуральными числами. Отметим, что не всякую целочисленную функцию можно связать с алгоритмом, поскольку алгоритм — это конечное предписание, а значит, их счетное число, а целочисленных функций — континуум. Значит, надо выделять специальный класс целочисленных функций, значение которых можно получить за конечное число элементарных действий. В литературе такие функции называют ***эффективно вычислимыми***. Это понятие, как и алгоритм, относится к интуитивным, первичным. Фактически, эффективно вычислимая функция — это алгоритм, но как бы с другой точки зрения.

Один из подходов в теории алгоритмов — представление алгоритма как теории эффективно вычислимых функций. Другой подход — представление алгоритма как процесса функционирования некоторой абстрактной машины. Исходный объект, представленный группой натуральных чисел, интерпретируется как исходной состояние абстрактной машины. В каждый момент времени (которое является дискретным) абстрактная машина из известного состояния пере-

ходит в другое состояние, строго определенное исходным состоянием. Некоторые состояния являются конечными. Достигнув такого состояния, машина „выплевывает“ результат.

Оба этих подхода являются ограничением исходного интуитивного понятия „алгоритм“. Есть несколько целей такого ограничения. Во-первых, общее понятие алгоритма слишком обширно, чтобы о нем можно было сказать что-то содержательное. (Аналогично что можно сказать о функции действительного переменного вообще? Чтобы получить какую-то теорию, мы ограничиваем этот слишком широкий класс функций, вводя понятие непрерывности, дифференцируемости и т.п.) Во-вторых, теория алгоритмов начиналась с решения вполне конкретных задач, в первую очередь задач математической логики и оснований математики. Для таких целей общее понятие алгоритма не нужно. Наконец, в-третьих, для решения проблем разрешимости формальных теорий, когда требуется установить отсутствие алгоритмов, решающих задачу, необходимо математически точное определение алгоритма, а это без ограничения понятия невозможно.

Исторически сложились три подхода к формальному определению алгоритма:

1) машины Тьюринга — Поста, предложенные Тьюрингом и независимо от него Постом в 1936 г.;

2) частично рекурсивные функции, предложенные Клини в 1936–37 гг.;

3) нормальные алгорифмы Маркова (1951 г.).

Все эти подходы объединяют общим понятием „вычислительная модель“. Указанные три вычислительные модели обладают важным свойством представительности. Это свойство заключается в том, что любой алгоритм со счетным множеством исходов X и со счетным множеством результатов Y может быть реализован в рамках данной вычислительной модели. Утверждение такого сорта не может быть доказано математическими средствами, поскольку общее понятие алгоритма выходит за рамки математики. Его истинность может быть лишь подтверждена практикой. Оно известно как ***тезис Черча***, который впервые был сформулирован для рекурсивных функций в форме: „любая эффективно вычислимая функция является частично рекурсивной“. Однако отметим, что эквивалентность вычислительных моделей может быть доказана математически строго.

В связи с понятием „алгоритм“ вспомним другое понятие „исчисление“, под которым в рамках математической логики имелось в виду логическое исчисление (т.е. исчисление математической логики). В общем исчисление представляет собой некий алфавит и набор правил, каждое из которых позволяет заменять то или иное слово другим. Процесс применения правил — это вывод. Вывод начинается с применения одного из правил с пустой посылкой, которые обычно формулируют в виде аксиом. Это действительно похоже на применение алгоритма, но отличается от алгоритма следующим. Если алгоритм носит повелительный характер (из данного слова мы можем перейти к другому только одним способом — детерминированность алгоритма), то исчисление носит разрешительный характер (для преобразования данного слова мы можем применять любое из возможных правил). Можно сказать, что алгоритмический процесс — это линейный вывод, в то время как вывод в исчислении носит древовидный характер. Связь между алгоритмами и исчислениями можно уловить из дальнейшего изложения.

8.2. Конструктивные объекты

Объекты с которыми работает алгоритм. Конструктивные объекты. Алфавит. Слова. Конкатенация. Подслова и вхождения.

Как отмечено, на объекты, которые перерабатываются алгоритмами, накладываются серьезные ограничения. В связи с этим возникает понятие ***конструктивный объект***, под которым понимается такой объект, который создан из конечного числа простейших объектов и в котором эти простейшие составляющие могут быть легко локализованы. Обычно считают, что простейших объектов конечное множество. Таким образом, конструктивный объект —

это конечный объект, т.е. представляет собой конечное множество. Однако не всякое конечное множество конструктивно. Требуется, чтобы по объекту можно было понять и историю его создания из простейших.

Например, само по себе натуральное число, как мера количества, не является конструктивным объектом. Однако его легко превратить в конструктивный, если рассматривать десятичную запись натурального числа. При этом простейшими объектами являются десятичные цифры. На самом деле можно ограничиться лишь одним символом: I —1, II — 2, III — 3 и т.д.

Как видим понятие конструктивного объекта легко укладывается в терминологию „буква“ — „алфавит“ — „слово“. При этом алфавит —- произвольное конечное множество, элементы которого мы называем буквами, слово — произвольный упорядоченный набор букв (***кортеж***).

Разумеется, в качестве букв, составляющих алфавит, мы будем выбирать не отвлеченные объекты (например, автомобили в г. Москва), а привычные языковые символы (буквы латинского, греческого алфавитов, кириллицы и др.). Порядок в конечном множестве можно рассматривать как установление взаимно-однозначного (биективного) соответствия между элементами множества и некоторым отрезком натурального ряда. Однако в данном случае проще считать понятия „меньше“ — „больше“ синонимами понятий „левее“ —„правее“ при записи кортежа. Единственное здесь уточнение: мы не будем использовать для записи кортежа разделителей, поскольку объекты, составляющие кортеж, неделимы.

Напомним, что существует формальный объект, называемый пустым кортежем, который есть пустое множество букв алфавита. Такое множество можно считать упорядоченным априори. Пустой кортеж (пустое слово) будем обозначать символом Λ.

Пример 8.4. Алфавит, содержащий единственную букву, например I, позволяет представить слова Λ, I, II, III, IIII, ... Эти слова можно отождествить с натуральными числами (при этом пустое слово будет соответствовать числу 0, которое в математической логике и теории алгоритмов считают натуральным). Введение дополнительного символа $-$ позволяет записывать и другие целые числа: $-$I, $-$II. Есть, правда и другие слова, например II$-$III, которые можно рассматривать как запись разности двух натуральных чисел. Но есть и такое слово: I$-$II$-$III. Как его рассматривать? Ему также можно придать смысл, установив порядок выполнения операций. Но есть и другой вариант действий: объявить, что не все слова являются правильными. В принципе это естественно: те слова, которые у нас будут возникать, будут порождаться определенными правилами. Некоторые слова, которые вообще-то состоят из букв данного алфавита, не могут появиться, поскольку для них нет порождающих правил. Если считать, что мы можем добавить к слову символ $-$ только если это слово пустое, а символ I можем добавлять только справа, то получим только „правильные“ записи целых чисел.

Пример 8.5. Терминология „буква“ — „алфавит“ — „слово“ уже встречалась в курсе математической логики. Однако там под буквой понималась переменная, а алфавит автоматически считался счетным множеством. На самом деле требование счетности с точки зрения конструктивности не обременительно. Формальную теорию можно было строить и с конечным алфавитом. Например, рассмотрим алфавит с буквами V, C, F, P, I, $\#$. Эти буквы позволяют порождать слова вида $VIII\#$, образующие уже новый счетный алфавит. Здесь первая буква определяет тип символа (в данном случае V — переменная), последующие символы — порядковый номер (в данном случае три), а последняя буква — признак конца этого микрослова. Эти микрослова можно соединять в любом порядке, но при этом всегда сохраняется возможность выделять эти образования в большом слове. В логико-математическом языке это дополнительное построение не вносит ничего нового, но усложняет теоретические построения.

Слова в данном алфавите A можно соединять. Если X и Y — слова, то под XY (или $X \cdot Y$ будем понимать слово полученное присоединением к слову X справа слова Y. Например, если $A = \{a, b\}$, $X = abb$, $Y = baba$, то $XY = abbbaba$. Такая операция часто называется конкатенацией (соединением). Мы используем для нее мультипликативную форму записи операции

(т.е. трактуем как умножение). Отсюда понятны обозначения $Y^0 = \Lambda$ (поскольку нейтральным элементом по конкатенации является пустое слово), $Y^k = Y^{k-1}Y$, $k = 1, 2, \ldots$

Отметим и другую терминологию, связанную с понятиями „алфавит“, „слово“ и уже встречавшуюся в курсе математической логики. Количество символов в слове X называется ***длиной слова*** X. Так, для алфавита A множество A^r представляет собой совокупность всех слов длины r. Объединение $\bigcup\limits_{r \geqslant 0} A^r$, обозначаемое как A^*, есть множество всех слов в алфавите A.

Если слово Y представимо в виде $Y = ZXW$, где Z и W могут быть и пустыми, то слово X называется ***подсловом*** слова Y. В частности, каждое слово является подсловом самого себя, а пустое слово является подсловом любого другого. У слова Y может быть несколько представлений вида $Y = Z_1XW_1 = Z_2XW_2 = \ldots$ Эти представления можно упорядочить по возрастанию длин подслов Z_1, Z_2 и т.д. При этом говорят о первом ***вхождении***, втором и т.п. слова X В слово Y. Так, первое вхождение пустого подслова — в самом начале слова.

Также говорят „слово X входит в слово Y“, имея ввиду, что X есть подслово слова Y. Если в представлении $Y = ZXW$ подслово Z пустое, т.е. на самом деле $Y = XW$, то подслово X называется ***началом слова*** Y. Аналогично вводится понятие ***конца слова***. Подслово (начало слова, конец) называется собственным, если оно не совпадает со всем словом и не является пустым.

Замечание 8.1. Слова — не единственный конструктивный объект, который можно построить с помощью данного алфавита. Другим простейшим конструктивным объектом являются деревья. Пусть задан алфавит A. Рассмотрим ориентированное дерево, вершины которого помечены буквами алфавита A, а дуги, исходящие из одной вершины, упорядочены, например маркировкой натуральными числами. Такое дерево назовем A-деревом. Максимальная степень исхода k по вершинам дерева может служить мерой сложности этого дерева. (A, k)-деревом называют любое A-дерево, максимальная степень исхода которого не превышает k.

Слова над алфавитом A можно рассматривать как $(A, 1)$-деревья. В то же время любое (A, k)-дерево можно по определенным правилам преобразовать в слово. Так что с теоретической точки зрения новые конструктивные объекты не дают ничего нового. #

Итак, объектами, над которыми работают алгоритмы, являются слова в некотором алфавите A. Результат работы алгоритма можно представить как своего рода правило, которое каждому слову из некоторого множества $X \subset A^*$ ставит в соответствие другое слово. Значит, с каждым алгоритмом можно связать отображение φ: $X \to A^*$, называемое ***словарной функцией***. Такая функция называется ***частичной***, если $X \neq A^*$, и ***полностью определенной***, если $X = A^*$.

Вообще говоря, естественно пробовать применять алгоритм (как отображение) к любому слову в данном алфавите. Тогда возможны три сценария завершения работы алгоритма. В стандартном сценарии мы получаем новое слово как результат работы алгоритма. Во втором сценарии мы на очередном шаге можем получить слово, к которому элементарные операции, используемые в алгоритме, не применимы. В этой ситуации работа алгоритма прекращается безрезультатно (например, деление на нуль в ряде численных алгоритмов). Мы можем модифицировать алгоритм, считая, что в данной ситуации он прекращает работу, а его результатом является некое заранее оговоренное слово (например ERROR). Наконец, возможен третий сценарий, когда алгоритм „зацикливается“. Простейший вариант такой ситуации — повторение результатов, получаемых на каждом шаге. Но такое повторение не обязательно.

Исходя из изложенного можно сказать, что алгоритм — это многократное применение некоторой словарной функции, начинающееся с некоторого исходного слова. Этот процесс останавливается, если на очередном шаге выполнено условие останова или слово, полученное на предыдущем шаге, вышло за пределы области определения словарной функции. Мы видим, что исследование алгоритмов по существу есть исследование словарных функций и решение задач представимости одних словарных функций с помощью других.

ОГЛАВЛЕНИЕ