Глава 5
Элементы теории алгоритмов

§ 34
Уточнение понятия алгоритма

Зачем нужно определение алгоритма?

Как вы знаете, **алгоритмом** называют точный набор инструкций для исполнителя, который приводит к решению задачи за конечное время.

Особый интерес проявляли к алгоритмам математики. Один из древнейших известных алгоритмов — алгоритм Евклида для вычисления наибольшего общего делителя (НОД) двух натуральных чисел. Само слово «алгоритм» (от имени математика IX века аль-Хорезми, которого считают основателем алгебры) ввёл в науку в XVII веке немецкий математик Г. В. Лейбниц.

Долгое время считалось, что для любой математической задачи можно найти метод (алгоритм) решения, просто для ряда задач такие алгоритмы ещё не найдены. Эту идею высказал аль-Хорезми, такой же точки зрения придерживались и другие математики вплоть до начала XX века.

Однако, несмотря на все усилия, решить некоторые задачи не удавалось в течение столетий. Например, безуспешно закончились многочисленные попытки найти алгоритм доказательства правильности любой теоремы на основе заданной системы аксиом.

В 1931 г. австрийский математик К. Гёдель доказал теорему о неполноте, смысл которой состоит в том, что в любой достаточно сложной формальной системе, основанной на аксиомах (например, в арифметике, где введены натуральные числа и операции сложения и умножения), есть утверждение, которое невозможно ни доказать, ни опровергнуть в рамках этой системы. Поэтому было высказано предположение о том, что некоторые задачи **алгоритмически неразрешимы**, т. е. для них в принципе не существует алгоритма решения, и поэтому искать его бессмысленно. Чтобы строго доказать или опровергнуть эту гипотезу, нужно было ввести математическое понятие алгоритма.

«Определение», которое мы привели в начале главы, часто называют *интуитивным*, потому что оно содержит такие «немате-

матические» понятия, как «точный набор», «инструкция», «исполнитель», «решение задачи». Эти термины невозможно записать строго, используя язык математики и логики, поэтому для математического доказательства такое определение не подходит.

Исследования в этой области, которые начали активно проводиться в 30-х годах XX века, привели к возникновению **теории алгоритмов**, которая занимается:

- доказательством алгоритмической неразрешимости задач;
- анализом сложности алгоритмов;
- сравнительной оценкой качества алгоритмов.

Значительный вклад в развитие теории алгоритмов внесли математики А. Тьюринг (Великобритания), Э. Пост (США), А. Чёрч (Великобритания), С. Клини (США) и А. А. Марков (СССР).

Что такое алгоритм?

Первые известные алгоритмы — это правила выполнения арифметических действий с числами. В них чётко определены объекты (числа в десятичной записи) и элементарные шаги (сложить, вычесть, перемножить два однозначных числа — вспомните таблицы сложения и умножения). Постепенно сложность задач, которые решались с помощью алгоритмов, увеличивалась, и понятие «шаг алгоритма» оказалось нечётким, размытым. Например, можно ли считать элементарным шагом разложение числа на простые множители или сложение многозначных чисел?

Со временем понятие алгоритма расширилось — сейчас мы говорим об алгоритмах для исполнителей, которые работают с текстами и другими объектами реального мира. Однако оказалось, что все эти объекты можно тем или иным способом закодировать в виде цепочек символов, так что любой алгоритм сводится к преобразованию одной символьной строки в другую. Таким способом можно представить и классические вычислительные алгоритмы — операции с цифрами. В алгоритме шахматной игры объекты — это фигуры на доске, но их расположение легко закодировать в символьной форме (вспомните запись шахматных партий).

Поэтому можно рассматривать только алгоритмы обработки символьных строк, а полученные результаты будут применимы к любым алгоритмам. Как вы знаете, текст, записанный с помощью любого алфавита, всегда можно перевести в двоичный код, поэтому, вообще говоря, достаточно рассматривать только алгоритмы, работающие с двоичными последовательностями.

Про любой алгоритм можно сказать следующее:

- алгоритм получает на вход дискретный объект (например, слово);
- алгоритм обрабатывает входной объект по шагам (дискретно), строя на каждом шаге промежуточные дискретные объекты; этот процесс может закончиться или не закончиться;
- если выполнение алгоритма заканчивается, его результат — это объект, построенный на последнем шаге;
- если выполнение алгоритма не заканчивается (алгоритм зацикливается) или заканчивается аварийно (например, в результате деления на 0), то результат его работы при данном входе не определён.

Любой алгоритм рассчитан на определённого исполнителя: он должен использовать только понятные этому исполнителю команды. Задание для исполнителя — это текст на специальном (формальном) языке, который обычно называют программой. Поэтому можно определить алгоритм как программу для некоторого исполнителя.

Напомним, что, с точки зрения теории алгоритмов, достаточно рассматривать только алгоритмы, работающие с цепочками символов, которые называют **словами** (рис. 5.1).

Рис. 5.1

Каждый алгоритм **задаёт (вычисляет) функцию**, которая преобразует входное слово в результат (выходное слово). Такая функция может быть не определена для некоторых входных слов, если алгоритм зацикливается.

Функция, заданная алгоритмом, может быть нигде не определена. Например, алгоритм

```
нц пока да
кц
```

зацикливается при любом входном слове.

Алгоритмы называются **эквивалентными**, если они задают одну и ту же функцию. То есть при любом входном слове оба алгоритма должны приводить к одному и тому же результату или зацикливаться (оба алгоритма не выдают никакого результата). Например, следующие алгоритмы для выбора минимального из значений переменных a и b эквивалентны:

```
если a<b то
  M:=a
иначе
  M:=b
все
```

```
M:=b
если a<b то
  M:=a
все
```

Универсальные исполнители

Как мы уже видели, понятие алгоритма оказывается «привязанным» к его исполнителю и некоторому языку программирования. Это не позволяет определить алгоритм как математический объект. Поэтому возникла идея попытаться построить **универсальный исполнитель.**

Универсальный исполнитель — это исполнитель, который может моделировать работу любого другого исполнителя.
Это значит, что для любого алгоритма, написанного для любого исполнителя, существует эквивалентный алгоритм для универсального исполнителя.

Такой исполнитель можно было бы использовать для доказательства разрешимости или неразрешимости задач. Если удаётся построить алгоритм решения задачи для универсального исполнителя, то задача разрешима. Если доказано, что алгоритм не существует, то задача неразрешима. Система команд такого исполнителя должна быть как можно проще — так его будет легче использовать в доказательствах.

В середине XX века разными учёными независимо друг от друга были предложены несколько исполнителей, претендующих на роль универсальных (они будут рассмотрены далее), причём в теории алгоритмов доказано, что все они эквивалентны друг другу. Это означает, что для любого алгоритма для одного уни-

версального исполнителя можно построить эквивалентный алгоритм для другого универсального исполнителя.

Как же связан универсальный исполнитель с проблемой строгого определения алгоритма?

Любой алгоритм может быть представлен как программа для универсального исполнителя.

Это основная идея теории алгоритмов. Строго доказать это утверждение невозможно, потому что здесь используется интуитивное понятие «алгоритм».

Как мы увидим, каждый универсальный исполнитель описывается с помощью математических терминов, поэтому на его основе можно дать строгое определение алгоритма:

Алгоритм — это программа для универсального исполнителя.

Универсальный исполнитель — это некоторая **модель вычислений**, которая задаёт способ описания алгоритмов и их выполнения. Модель вычислений должна содержать:

- «процессор», задающий систему команд и способ их выполнения;
- «память», определяющую способ хранения данных;
- язык программирования (способ записи программ);
- способ ввода данных (чтения входного слова);
- способ вывода слова-результата.

Все универсальные исполнители эквивалентны, поэтому последнее приведённое определение алгоритма не зависит от конкретного исполнителя.

Машина Тьюринга

Первым предложил универсальный исполнитель английский математик Алан Тьюринг. Придуманное им воображаемое устройство состоит из трёх частей:

- *бесконечной ленты*, разделённой на ячейки;
- *каретки* (читающей и записывающей головки);
- *программируемого автомата*.

А. Тьюринг
(1912–1956)

Программируемый автомат управляет кареткой, посылая ей команды в соответствии с заложенной в него сменяемой программой. Лента выполняет роль памяти компьютера, автомат — роль процессора, а каретка служит для ввода и вывода данных. Такое устройство называют **машиной Тьюринга**.

Теоретически лента в машине Тьюринга бесконечна, однако в каждый момент времени работы машины используется лишь конечная её часть.

Каретка в любой момент времени находится над одной ячейкой, автомат может читать и изменять содержимое этой ячейки, которая называется **текущей (рабочей) ячейкой** (рис. 5.2).

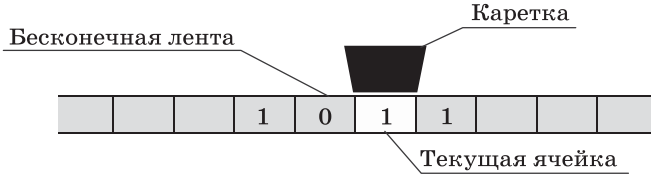

Рис. 5.2

В каждую ячейку ленты можно записать один любой символ, принадлежащий выбранному алфавиту. Любой алфавит обязательно содержит пробел (пустой символ, соответствующий «чистым» участкам ленты), который мы будем обозначать знаком ☐. Алфавит обычно обозначается буквой A, а его элементы — строчными буквами a с индексами: $A = \{a_1, a_2, \ldots, a_N\}$. Например, алфавит машины Тьюринга, работающей с двоичными числами, задаётся в виде $A = \{0, 1, ☐\}$.

Непрерывную цепочку символов на ленте называют **словом**. На рисунке 5.2 лента содержит слово «1011», которое можно воспринимать как двоичное число.

Автоматом называют устройство, работающее без участия человека. Автомат в машине Тьюринга имеет несколько **состояний** и при определённых условиях переходит из одного состояния в другое. Состояние автомата определяет ту промежуточную задачу,

которую решает автомат в данный момент. Это напоминает состояния человека: ночью он спит (состояние 1), утром встаёт и умывается (состояние 2), завтракает (состояние 3), идёт на работу (состояние 4) и т. д.

Множество всех состояний автомата обозначается буквой Q, а его элементы — строчными буквами q с индексами: $Q = \{q_1, q_2, ..., q_M\}$. Принято, что в начальный момент машина Тьюринга находится в состоянии q_1.

Особое состояние q_0 — это **состояние останова**. Если машина переходит в это состояние, выполнение программы сразу останавливается.

Автомат управляется **программой**. Во время каждого шага программы автомат выполняет последовательно три действия:

1) изменяет символ в рабочей ячейке на другой (или оставляет без изменений);
2) перемещает каретку влево или вправо (или оставляет на месте);
3) переходит в другое состояние (или остаётся в прежнем состоянии).

Поэтому при составлении программы для каждой пары (*символ, состояние*) нужно определить три параметра: символ a_i из выбранного алфавита A, направление перемещения каретки ($\leftarrow$ — влево, $\rightarrow$ — вправо, точка — нет перемещения) и новое состояние автомата q_k. Например, команда $1 \leftarrow q_2$ обозначает «заменить символ на 1, переместить каретку влево на 1 ячейку и перейти в состояние q_2».

Пример 1. На ленте записано число в двоичной системе счисления. Каретка находится где-то над числом. Требуется увеличить число на единицу.

Для того чтобы построить машину Тьюринга, нужно:

- определить алфавит машины Тьюринга A;
- выделить простейшие подзадачи и определить набор возможных состояний Q; задать начальное состояние q_1 и конечное состояние q_0 (в котором машина останавливается);
- составить программу, т. е. для каждой пары (a_i, q_k) определить команду, которую должен выполнить автомат.

Как мы уже выяснили, алфавит машины Тьюринга, работающей с двоичными числами, включает символы 0, 1 и пробел: $A = \{0, 1, \square\}$. Определим возможные состояния (разобьём задачу на элементарные подзадачи):

1) q_1 — автомат ищет правый конец слова (числа) на ленте;
2) q_2 — автомат увеличивает число на 1, проходя его справа налево, и останавливается, закончив работу.

Теперь займёмся программой. На первом этапе, когда автомат ищет конец слова, его работа может быть описана так:

1) если в рабочей ячейке записана цифра 0, переместиться вправо;
2) если в рабочей ячейке записана цифра 1, переместиться вправо;
3) если в рабочей ячейке пробел, переместить каретку влево и перейти в состояние q_2.

Тогда действия автомата в состоянии q_1 можно представить в виде таблицы, где в заголовках строк записываются символы алфавита, а в заголовках столбцов — состояния (рис. 5.3).

	q_1
0	$0 \to q_1$
1	$1 \to q_1$
□	$□ \leftarrow q_2$

Рис. 5.3

Заметим, что во всех случаях символ под кареткой не меняется. Кроме того, состояние меняется только в последней ячейке. Поэтому для упрощения записи не будем указывать в таблице то, что остаётся без изменений. Так, на наш взгляд, более кратко и понятно (рис. 5.4).

	q_1
0	$\to$
1	$\to$
□	$\leftarrow q_2$

Рис. 5.4

Второй этап — увеличение двоичного числа на единицу. Это можно сделать следующим способом (вспомните тему «Системы счисления»):

1) если в рабочей ячейке записана цифра 0, записать в неё 1 и стоп;
2) если в рабочей ячейке записана цифра 1, выполнить перенос в старший разряд — записать в ячейку 0 и переместиться влево;
3) если в рабочей ячейке пробел, записать в неё 1 и стоп.

Для того чтобы остановить работу машины Тьюринга, нужно перевести её в состояние останова q_0. Теперь можно добавить в таблицу столбец, соответствующий состоянию q_2 (рис. 5.5).

	q_1	q_2
0	$\rightarrow$	$1\ .\ q_0$
1	$\rightarrow$	$0\ \leftarrow$
$\square$	$\leftarrow\ q_2$	$1\ .\ q_0$

Рис. 5.5

Построенная полная таблица (см. рис. 5.5) — это и есть **программа для машины Тьюринга**. Обратите внимание, что мы разбили исходную задачу на подзадачи, для каждой из них составили программу, а потом их соединили. Две подзадачи связаны через ячейку ($\square$, q_1), в которой состояние автомата изменяется на q_2. В данном простейшем случае в каждом из двух алгоритмов было использовано только одно состояние, но это не обязательно — можно таким же способом соединять и более сложные алгоритмы. Если алгоритмы A и $Б$ можно запрограммировать на машине Тьюринга, то и любую их комбинацию тоже можно запрограммировать.

Тьюринг предположил, что:

> Любой алгоритм (в интуитивном смысле этого слова) может быть представлен как программа для машины Тьюринга.

Это утверждение в теории алгоритмов известно как **тезис Чёрча–Тьюринга**.

Машина Тьюринга может быть строго задана с точки зрения математики. Алфавит A и набор возможных состояний Q могут быть записаны в виде множеств, а программа — в виде пятёрок вида $(a_i, q_k, a_j, d_{ik}, q_m)$, задающих команду «если машина находится в состоянии q_k и в рабочей ячейке записан символ a_i, то записать в рабочую ячейку символ a_j, сместиться в направлении d_{ik} и перейти в состояние q_m». Например, приведённая выше программа увеличения двоичного числа на 1, записанная в виде таких пятёрок, выглядит так:

$(0, q_1, 0, \rightarrow, q_1)$, $(1, q_1, 1, \rightarrow, q_1)$, $(\square, q_1, \square, \leftarrow, q_2)$,
$(0, q_2, 1, ., q_0)$, $(1, q_2, 0, \leftarrow, q_2)$, $(\square, q_2, 1, ., q_0)$.

Эта машина — математический объект, и данное на её основе определение алгоритма может использоваться для доказательств. Едва ли можно применить машину Тьюринга для решения практических задач, но эта простая модель алгоритма очень удобна для проведения теоретических исследований. В отличие от интуитивного определения алгоритма новое определение не содержит таких неопределённых понятий, как «инструкция», «исполнитель», «решение задачи». Таким образом, формальное определение слова «алгоритм» (по Тьюрингу) выглядит так: алгоритм — это программа для машины Тьюринга.

Машина Поста

Э. Л. Пост
(1897–1954)

Практически одновременно с Тьюрингом (в том же 1936 г.) и независимо от него американский математик Э. Л. Пост предложил ещё более простую систему обработки данных, на основе которой позднее была построена так называемая **машина Поста**.

Лента в машине Поста (так же как и в машине Тьюринга) бесконечна и разбита на ячейки. Каждая ячейка может содержать метку (быть отмечена) или не содержать её (пустая ячейка) (рис. 5.6).

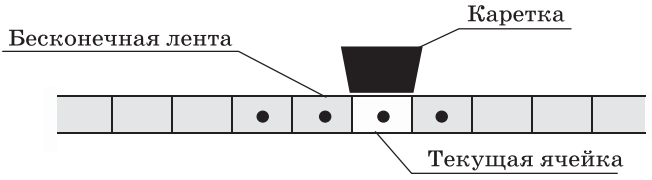

Рис. 5.6

Таким образом, Пост сократил алфавит всего до двух цифр. Это допустимо, потому что любые данные можно перекодировать в двоичный код, сопоставив каждой букве исходного алфавита уникальную последовательность нулей и единиц.

Кроме того, алгоритм работы машины Поста задаётся не в виде таблицы, а как программа, состоящая из отдельных команд. Система команд машины Поста содержит только 6 команд:

← — переместить каретку на 1 ячейку влево;
→ — переместить каретку на 1 ячейку вправо;
0 — стереть метку в рабочей ячейке (записать 0);

[. . .]