П.Г. Ключарев, Д.А. Жуков

ВВЕДЕНИЕ В ТЕОРИЮ АЛГОРИТМОВ

Московский государственный технический университет
имени Н.Э. Баумана

П.Г. Ключарев, Д.А. Жуков

ВВЕДЕНИЕ В ТЕОРИЮ АЛГОРИТМОВ

Рекомендовано
Научно-методическим советом МГТУ им. Н.Э. Баумана
в качестве учебного пособия

Москва
Издательство МГТУ им. Н.Э. Баумана
2012

УДК 510.5(075.8)
ББК 22.12
К52

Рецензенты: *А.Н. Велигура, А.Ю. Голубков*

Ключарев П. Г.

К52 Введение в теорию алгоритмов : учеб. пособие / П.Г. Ключарев, Д.А. Жуков. – М.: Изд-во МГТУ им. Н.Э. Баумана, 2012. – 37, [3] с.: ил.

Рассмотрены машины Тьюринга, вопросы алгоритмической разрешимости, основные классы сложности, *NP*-полнота, схемная сложность.

Для студентов МГТУ им. Н.Э. Баумана, обучающихся по специальностям «Информационная безопасность автоматизированных систем» и «Компьютерная безопасность». Пособие может быть полезно студентам других специальностей, связанных с информатикой, вычислительной техникой и информационной безопасностью.

УДК 510.5(075.8)
ББК 22.12

ВВЕДЕНИЕ

В любой области науки и техники и просто в повседневной жизни человек постоянно сталкивается с решением различных задач. Примерами задач могут быть подсчет стоимости продуктов для банкета, выбор оптимального маршрута для поездки в другой конец города, вычисление квадратного корня из заданного числа, расчет оптимальной траектории космического аппарата, шифрование файла.

Решая любую конкретную задачу, человек выполняет некоторую последовательность действий. Формальное описание такой последовательности действий называют алгоритмом. Несмотря на то что алгоритмы могут быть предназначены для решения совершенно разных задач и быть совсем непохожими друг на друга, существуют свойства, характерные для всех алгоритмов.

Общие свойства алгоритмов изучает теория алгоритмов, которой и посвящено настоящее учебное пособие. Теория алгоритмов представляет собой область дискретной математики, находящуюся на стыке математической логики и информатики. Изучение общих свойств задач и алгоритмов с математической точки зрения имеет огромное значение как для чистой математики, так и для большинства областей информатики, прежде всего для программирования и информационной безопасности.

Одним из краеугольных камней информационной безопасности является криптография. Многие методы криптографии основаны на тех или иных свойствах алгоритмов. В частности, почти все современные криптографические алгоритмы и протоколы базируются на однонаправленных функциях, т. е. на функциях, которые можно быстро вычислить, но для обращения которых требуется очень много времени.

В учебном пособии приведены основные понятия теории алгоритмов. Эта теория непрерывно развивается и накопила знания, далеко выходящие за рамки этого пособия. Заинтересованный читатель найдет много дополнительных сведений в книгах, приведенных в списке литературы [1–16].

1. ОСНОВНЫЕ ПОНЯТИЯ

Учебное пособие рассчитано на читателя, который освоил теорию булевых функций, теорию графов и теорию вероятностей. Читателю, не изучавшему эти разделы математики, мы рекомендуем ознакомиться с ними, используя, например, работы [6, 11, 15].

Далее мы будем применять некоторые стандартные обозначения, в частности:

$\mathbb{N} = \{1; 2; 3; ...\}$ – множество натуральных чисел;

$\mathbb{R}$ – множество действительных чисел.

Сначала введем основные термины и определения.

Определение 1. Рассмотрим две функции: $f, g : \mathbb{N} \to \mathbb{N}$. Формула $f = O(g)$ означает, что существуют такие $c \in \mathbb{R}$ и $n_0 \in \mathbb{N}$, что при любом $n > n_0$ выполняется $f(n) \le cg(n)$.

Определение 2. Формула $f = \Omega(g)$ означает, что $g = O(f)$.

Определение 3. Формула $f = \Theta(g)$ означает, что $f = O(g)$ и $g = O(f)$.

Определение 4. Назовем функцию $f : \mathbb{N} \to \mathbb{N}$ полиномиальной, если существует такое $k \in \mathbb{R}$, что $f(n) = O(n^k)$.

Дадим теперь более строгое определение понятия задачи, с тем чтобы это понятие можно было теоретически исследовать.

Определение 5. Назовем *абстрактной задачей* некоторое бинарное отношение между элементами двух множеств: множества условий и множества решений.

Примером является задача об умножении двух чисел: условие – пара чисел, а решение – одно число.

Определение 6. Задачи, у которых множество решений состоит из двух элементов (например, ДА и НЕТ), будем называть *задачами распознавания*.

Примером задачи распознавания может служить следующая задача: определить, является ли данный граф гамильтоновым.

Понятие задачи тесно связано с понятием языка.

Определение 7. *Алфавитом* называется конечное непустое множество символов.

Часто используется алфавит $\{0;1\}$, называемый *двоичным*, или *бинарным*, алфавитом.

Определение 8. *Словом* над алфавитом A называется набор символов из алфавита A.

Определение 9. *Языком* над алфавитом A называется множество слов над алфавитом A.

Каждой задаче распознавания можно поставить в соответствие язык. Слово принадлежит этому языку тогда и только тогда, когда для этого входного слова ответ задачи – ДА. В то же время любому языку можно поставить в соответствие задачу распознавания этого языка. В связи с этим мы будем употреблять термины «язык» и «задача распознавания» как синонимы.

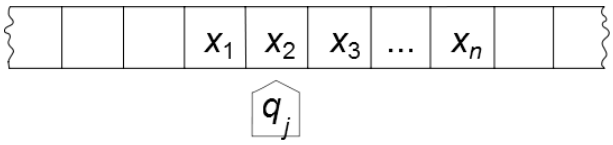
В качестве примера языка можно привести язык над алфавитом $\{0; 1\}$, состоящий из записей всех простых чисел в двоичной системе счисления. Соответствующая задача распознавания формулируется так: является ли данное число простым?

Условие задачи можно некоторым разумным образом закодировать над конечным алфавитом (следовательно, и над двоичным алфавитом). Например, рациональное число всегда можно закодировать над двоичным алфавитом, используя представление с плавающей точкой. Граф можно закодировать с помощью матрицы смежности, матрицы инцидентности или списка вершин. Аналоговый сигнал можно закодировать в виде последовательности отсчетов, взятых с некоторым периодом дискретизации, каждый из которых закодирован в виде двоичного числа. Конкретный способ кодирования следует выбирать, исходя из специфики задачи.

Во введении мы говорили об алгоритме как о некоторой последовательности действий. Такое определение нельзя признать удовлетворительным. Действительно, совершенно непонятно, что такое «действия» и кто их выполняет. Поэтому первой задачей теории алгоритмов является строгое определение понятия алгоритма. Делается это путем постулирования некоторой конструкции, удовлетворяющей интуитивным представлениям об алгоритме. Такая конструкция называется *алгоритмической моделью*. Было предложено большое число различных алгоритмических моделей. С одной из таких моделей – машиной Тьюринга – мы и познакомимся.

Определение 10. *Машина Тьюринга* состоит из *головки* и *ленты*, по которой эта головка перемещается (см. рисунок). Лента

бесконечна в обе стороны и разделена на ячейки, причем каждая ячейка хранит символ из некоторого конечного алфавита A. У машины Тьюринга имеется внутреннее состояние, которое выбирается из некоторого конечного множества Q. Некоторые состояния выделяются особо и называются допускающими. Головка машины Тьюринга в любой момент времени обозревает какую-либо одну клетку ленты.



Машина Тьюринга

Работает машина Тьюринга по шагам. В начальный момент времени на ленте записан *вход*, представляющий собой конечный набор символов, а головка обозревает ячейку, содержащую первый (крайний левый) символ из этого набора. Все остальные ячейки ленты содержат пробелы. На каждом шаге с помощью некоторой заданной *функции переходов* исходя из символа, записанного в обозреваемой головкой ячейке, и текущего состояния выбираются символ, записываемый в обозреваемую клетку, новое состояние и направление перехода головки. Направлений перехода в нашей формализации будет три:

1) влево (L) – головка сдвигается на одну клетку влево;

2) вправо (R) – головка сдвигается на одну клетку вправо;

3) на месте (S) – головка остается там же, где была.

Машина Тьюринга завершает работу в двух случаях:

1) если очередное состояние принадлежит заранее определенному подмножеству множества состояний – множеству допускающих состояний F. В этом случае говорят, что машина Тьюринга остановилась в допускающем состоянии;

2) если функция переходов не определена на аргументе, соответствующем символу обозреваемой клетки и текущему состоянию. В этом случае говорят, что машина Тьюринга остановилась в недопускающем состоянии.

Более формально машиной Тьюринга называется набор

$$MT = \left(A, Q, F, q_0, \delta\right), \tag{1}$$

где A – алфавит, причем он обязательно содержит пробельный символ; Q – конечное множество состояний; F – множество допускающих состояний, причем $F \subseteq Q$; q_0 – начальное состояние, $q_0 \in Q$; δ – частично определенная функция переходов машины Тьюринга:

$$\delta : A \times Q \to A \times Q \times \{L, S, R\}. \quad (2)$$

Функцию δ удобно задавать в виде таблицы, строки которой соответствуют состояниям, а столбцы – символам алфавита. В каждой ячейке таблицы задан символ, который машина записывает на ленту, состояние, в которое она переходит, и направление движения головки. Такую таблицу будем называть *таблицей переходов*.

В качестве примера построим машину Тьюринга, прибавляющую единицу к данному двоичному числу. Для того чтобы это сделать, надо сначала подвести головку машины к крайнему правому символу (это будет производиться в состоянии q_0), затем прибавить единицу (это будет происходить в состоянии q_1) и далее перейти в допускающее состояние q_f. Таблица переходов этой машины Тьюринга приведена ниже (табл. 1).

Таблица 1

Таблица переходов машины Тьюринга, прибавляющей единицу к заданному числу

Состояние	Символ		
	0	1	_
q_0	0, q_0, R	1, q_0, R	_, q_1, L
q_1	1, q_f, S	0, q_1, L	1, q_f, S
$q_f \in F$			

Теперь построим машину Тьюринга, допускающую язык $\{0^n 1^n \mid n \geq 1\}$. На ее вход подается слово над алфавитом $\{0; 1\}$. Начиная с левого конца входного слова, машина Тьюринга заменяет символ «0» символом «*A*», затем передвигается вправо, пока не достигнет символа «1». Его она заменяет символом «*B*» и движется

влево до символа «*A*», далее передвигается вправо и начинает сначала. Работа заканчивается в допускающем состоянии, если символов «0» и «1» не осталось после очередной итерации цикла, и в недопускающем состоянии, если остались нули, но не осталось единиц либо, наоборот, остались единицы, но не осталось нулей. Таблица переходов этой машины Тьюринга приведена ниже (табл. 2).

Таблица 2

Таблица переходов машины Тюринга, допускающей язык $\{0^n1^n \mid n \geq 1\}$

Состояние	Символ				
	0	1	*A*	*B*	_
q_0	A, q_1, R			B, q_3, R	
q_1	$0, q_1, R$	B, q_2, L		B, q_1, R	
q_2	$0, q_2, L$		A, q_0, R	B, q_2, L	
q_3				B, q_3, R	$\_, q_f, R$
$q_f \in F$					

Очень сложно описывать реальные алгоритмы с помощью машины Тьюринга. Однако этого и не требуется. Машина Тьюринга предназначена для теоретических исследований. Ценность машины Тьюринга заключается в том, что для нее считается справедливым следующий тезис.

Тезис Черча – Тьюринга. Любой алгоритм (в интуитивном смысле) может быть реализован в виде машины Тьюринга.

Тезис Черча – Тьюринга означает, что машина Тьюринга является определением понятия «алгоритм». Поэтому в дальнейшем мы будем отождествлять эти понятия.

Тезис Черча – Тьюринга не является теоремой и не может быть доказан, поскольку никакого лучшего определения понятия «алгоритм» в настоящее время не существует. В то же время имеется достаточно большое количество других алгоритмических моделей, таких, как машина Поста, лямбда-исчисление, нормальные алгоритмы Маркова и др. Доказано, что все они эквивалентны машине Тьюринга в том смысле, что любой алгоритм, который может быть реализован в одной алгоритмической модели, может быть реализован и в другой, а если какая-либо задача не может быть решена в

одной алгоритмической модели, она не может быть решена и в другой.

Кроме того, машина Тьюринга может имитировать любой современный компьютер. Однако компьютер может имитировать далеко не любую машину Тьюринга. Связано это с тем, что в отличие от компьютера машина Тьюринга обладает бесконечной памятью в виде бесконечной ленты.

Определение 11. Назовем машину Тьюринга применимой к некоторому слову *w*, если она, получив на входе это слово, завершит работу, выполнив конечное число шагов.

Определение 12. Будем говорить, что машина Тьюринга допускает слово *w*, если, получив на вход это слово, она останавливается в допускающем состоянии, совершив конечное число шагов.

Определение 13. Будем говорить, что машина Тьюринга не допускает слово *w*, если, получив на вход это слово, она останавливается в недопускающем состоянии, выполнив конечное число шагов.

Машина Тьюринга, как и любой дискретный объект, может быть представлена в виде *кода*, т. е. слова над конечным алфавитом. Это можно сделать различными способами. Далее мы будем использовать простой, но не слишком эффективный способ кодирования.

Перенумеруем все состояния машины Тьюринга $Q = \{q_1, q_2, ..., q_t\}$ так, чтобы номера состояний были натуральными числами, а множество допускающих состояний состояло из состояний с наибольшими номерами: $F = \{q_{t-u+1}, ..., q_t\}$, где u – число допускающих состояний. Начальным состоянием объявим состояние q_1. Пронумеруем направления движения головки машины: $L - 1; R - 2; S - 3$.

Пусть $\delta(q_i, a_j) = (q_k, a_l, d_m)$, где d_m – номер направления движения головки. Закодируем каждый элемент таблицы переходов в виде двоичного слова $1^i01^j01^k01^l01^m$, если функция переходов для этих значений аргумента определена, и в виде двоичного слова 1, если она не определена. В результате образуется конечная последовательность таких кодов: $c_1, c_2, ..., c_r$. Остается конкатенировать все эти последовательности, разделив их специальными разделителями и предварив информацией о мощности алфавита,

о мощности множества состояний и о мощности множества допускающих состояний. В результате получим следующую запись:

$$\text{code}(M) = 1^{|A|}01^{|Q|}01^{|F|}00c_1 00c_2 00...00c_r. \qquad (3)$$

Итак, мы сопоставили каждой машине Тьюринга код. По машине Тьюринга всегда можно определить код, а по коду – машину Тьюринга.

Отметим, что если у нас есть слово w над двоичным алфавитом, то мы можем сопоставить этому слову натуральное число, просто приписав к нему слева единицу: $1w$. Пользуясь этим методом, можно каждой машине Тьюринга сопоставить натуральное число – номер машины Тьюринга. При этом если некоторому номеру не соответствует ни одна машина Тьюринга, то будем считать, что ему соответствует машина Тьюринга, которая независимо от входа останавливается в недопускающем состоянии.

Итак, мы ввели нумерацию машин Тьюринга. Машине Тьюринга мы всегда можем сопоставить некоторый номер, а по произвольному натуральному числу – получить машину Тьюринга, ему соответствующую.

2. АЛГОРИТМИЧЕСКИ НЕРАЗРЕШИМЫЕ ЗАДАЧИ

Пусть дана некоторая задача. Встает закономерный вопрос: можно ли эту задачу решить? Имеются три возможности:

1) задачу решить можно и алгоритм решения известен. Такие задачи называют алгоритмически разрешимыми;

2) задачу решить можно, но алгоритм решения пока не известен. Такие задачи также являются алгоритмически разрешимыми, несмотря даже на то, что решить их нет возможности. Важной является принципиальная возможность решения;

3) задачу решить нельзя, т. е. алгоритм решения не может быть найден. Такие задачи называются алгоритмически неразрешимыми.

Оказывается, далеко не для любой задачи существует алгоритм решения. В этом разделе мы докажем существование алгоритмически неразрешимых задач.

Определение 14. Назовем задачу *алгоритмически разрешимой*, если существует машина Тьюринга, решающая эту задачу. Если такой машины Тьюринга не существует, будем называть задачу *алгоритмически неразрешимой*.

Теорема 1. Существует алгоритмически неразрешимая задача.

Доказательство. Используем тот факт, что, как было показано выше, все двоичные слова можно пронумеровать. Каждой машине Тьюринга поставим в соответствие бесконечную последовательность, *j*-й элемент которой будет равен единице, если она допускает *j*-е слово, и будет равен нулю – если не допускает. Назовем такую последовательность *характеристической последовательностью* машины Тьюринга. Такая последовательность полностью определяет язык, допускаемый машиной Тьюринга.

Объединим характеристические последовательности машин Тьюринга в бесконечную матрицу *M*. В этой матрице элемент M_{ij} равен единице, если *i*-я машина Тьюринга допускает *j*-е слово, и равен нулю – в противном случае.

Рассмотрим теперь язык L_d, определенный характеристической последовательностью $\overline{M}_{11}, \overline{M}_{22}, \overline{M}_{33}, \ldots$, состоящей из отрицаний элементов, находящихся на главной диагонали матрицы *M*. Такой язык называется *языком диагонализации*. Этот язык не совпадает ни с одним языком из матрицы *M*, и, следовательно, не существует машины Тьюринга, которая бы его допускала.

Приведем теперь явный пример алгоритмически неразрешимой задачи.

Определение 15. Назовем машину Тьюринга *самоприменимой*, если она применима к своему коду.

Определение 16. *Задача о самоприменимости* формулируется следующим образом: распознать, самоприменима ли данная машина Тьюринга.

Теорема 2. Задача о самоприменимости алгоритмически неразрешима.

Доказательство. Предположим, что эта задача алгоритмически разрешима. Тогда существует машина Тьюринга *M*, которая в качестве входа получает код машины Тьюринга *x*. Причем, если машина Тьюринга *x* самоприменима, машина Тьюринга *M* останавливается в допускающем состоянии, а если машина Тьюринга *x* несамоприменима, то машина Тьюринга *M* останавливается в недопускающем состоянии.

Изменим машину Тьюринга *M* таким образом, что она будет зацикливаться на входах, соответствующих самоприменимым машинам Тьюринга. Выполнить это просто. Достаточно все допус-

кающие состояния объявить недопускающими и сделать так, чтобы машина Тьюринга M в них зацикливалась (например, сдвигала головку вправо на всех символах алфавита, оставаясь в том же состоянии). Назовем эту новую машину M_1.

Является ли машина Тьюринга M_1 самоприменимой? Предположим, что это так, тогда очевидно, что она зацикливается, получив саму себя на вход; следовательно, она не является самоприменимой. Предположим теперь, что она несамоприменима. В этом случае согласно построению она, получив свой код на вход, остановится (в недопускающем состоянии) и, следовательно, является самоприменимой.

Мы пришли к противоречию, поэтому, построить машину Тьюринга M невозможно, что и требовалось доказать.

Итак, мы привели примеры двух алгоритмически неразрешимых задач (см. теоремы 1 и 2). Теперь у нас появился новый способ доказательства алгоритмической неразрешимости – сведéние. Способ состоит в следующем: пусть мы хотим доказать алгоритмическую неразрешимость некоторой задачи A. Для доказательства этого достаточно показать, что если бы существовал алгоритм решения задачи A, то можно было бы решить и задачу B, для которой алгоритмическая неразрешимость уже установлена. Чтобы проиллюстрировать этот подход, докажем алгоритмическую неразрешимость еще одной задачи.

Определение 17. *Задача об останове* формулируется следующим образом. Даны машина Тьюринга M и слово w. Распознать, завершит ли машина Тьюринга M свою работу, если на ее вход подать слово w.

Теорема 3. Задача об останове алгоритмически неразрешима.

Доказательство. Предположим, что задача об останове алгоритмически разрешима. В этом случае существует машина Тьюринга M_s, которая, получив на вход два слова (M и w), определит, остановится ли машина Тьюринга, закодированная словом M, на слове w. Если мы подадим на ее вход слова M и M, т. е. запишем два одинаковых слова, кодирующих некоторую машину Тьюринга M, то машина Тьюринга M_s распознает применимость машины Тьюринга M к самой себе. Однако в теореме 2 мы доказали, что такая задача алгоритмически неразрешима. Таким образом, мы пришли к противоречию. Следовательно, машина Тьюринга M_s существовать не может, что и требовалось доказать.

Заметим, что задача о распознавании языка диагонализации такова, что не существует машины Тьюринга, которая допускала бы этот язык. Ситуации с задачей о самоприменимости и с задачей об останове несколько иные: можно построить машину Тьюринга, которая допускала бы входы, соответствующие самоприменимым машинам Тьюринга (для этого достаточно смоделировать работу такой машины Тьюринга), но в то же время построить машину Тьюринга, которая останавливалась бы на прочих входах, невозможно. В связи с этим все языки можно подразделить на три класса: рекурсивные, рекурсивно-перечислимые и нерекурсивные.

Определение 18. Рекурсивным называется язык, соответствующий алгоритмически разрешимой задаче.

Определение 19. Рекурсивно-перечислимым называется не являющийся рекурсивным язык, для которого существует машина Тьюринга, допускающая лишь принадлежащие ему слова, к прочим же словам не применимая.

Определение 20. Нерекурсивным называется язык, не являющийся ни рекурсивным, ни рекурсивно-перечислимым.

Сколько же существует алгоритмически неразрешимых задач? Ответ на этот вопрос даст следующая теорема.

Теорема 4. Множество алгоритмически неразрешимых задач распознавания имеет мощность континуума, а множество алгоритмически разрешимых задач распознавания является счетным.

Доказательство. Вспомнив, что с каждой задачей распознавания ассоциирован язык, найдем мощность множества всех задач распознавания как мощность множества всех языков над конечным алфавитом. Поскольку множество слов счетно, множество всех языков как множество всех подмножеств счетного множества имеет мощность континуума. Множество различных машин Тьюринга как множество слов конечной длины счетно. Следовательно, множество алгоритмически разрешимых задач также счетно, в то время как множество алгоритмически неразрешимых задач имеет мощность континуума.

Итак, оказывается, что множество алгоритмически неразрешимых задач гораздо шире множества задач алгоритмически разрешимых. Тем не менее достаточно сложно привести пример алгоритмически неразрешимой задачи. Мы пока нашли только две такие задачи: задачу о самоприменимости (см. теорему 2) и задачу об останове (см. теорему 3). Перейдем теперь к теореме Райса, позволяющей доказывать неразрешимость более широкого класса задач.

Определение 21. *Свойством* рекурсивно-перечислимых языков называется некоторое множество рекурсивно-перечислимых языков.

Определение 22. Свойство рекурсивно-перечислимых языков называется *тривиальным*, если ему принадлежат все рекурсивно-перечислимые языки, или оно является пустым.

Например, свойство языка быть регулярным с формальной точки зрения представляет собой множество всех регулярных языков. Чтобы распознать какое-либо свойство языка, необходимо рассматривать машину Тьюринга, получающую на вход некоторое описание этого языка. Но невозможно подать на вход бесконечный язык в виде перечисления слов, поскольку на входе машины Тьюринга должно быть конечное слово. Поэтому нам надо выбрать способ конечного описания языка. Таким описанием может служить код машины Тьюринга, распознающей этот язык.

Теорема 5 (*теорема Райса*). Всякое нетривиальное свойство рекурсивно-перечислимых языков неразрешимо.

Доказательство. Пусть F_a – язык, допускаемый машиной Тьюринга с кодом a, и Q – машина Тьюринга, распознающая некоторое нетривиальное свойство. Будем считать, что язык машины Тьюринга, которая не останавливается независимо от входа, т. е. пустой язык, не принадлежит этому свойству. Это предположение не ограничивает общность, так как, если оно не выполняется для некоторого свойства, достаточно рассмотреть его дополнение. Поскольку машина Тьюринга Q распознает нетривиальное свойство, существует машина Тьюринга b, такая, что машина Тьюринга Q допускает ее код. Теперь мы рассмотрим машину Тьюринга H, получающую на вход пару (a, i). Она выполняет следующие действия:

1) вычисляет код t, соответствующий машине Тьюринга T, такой, что машина Тьюринга T сначала производит имитацию работы машины Тьюринга с кодом a на входном слове i, а затем производит имитацию работы машины Тьюринга с кодом b на входном слове i и возвращает результат;

2) моделирует работу машины Тьюринга Q на слове t, после чего останавливается в допускающем состоянии, если машина Тьюринга Q допускает t, и в недопускающем состоянии – в противном случае.

Отметим, что если на вход машины Тьюринга H подать код неприменимой к слову i машины Тьюринга, то машина Тьюринга T никогда не останавливается и машина Тьюринга Q не допускает

слово t. Однако если на вход машины Тьюринга H подать код применимой к слову i машины Тьюринга, то машина Тьюринга T проверяет принадлежность слова i к языку F_b, а машина Тьюринга Q допускает слово t.

Таким образом, машина Тьюринга H решает задачу об остановке. Это противоречит доказанной выше теореме 3 о том, что задача об останове алгоритмически неразрешима. Поэтому существование машины Тьюринга H невозможно.

3. КЛАССЫ СЛОЖНОСТИ

Определение 23. Будем говорить, что машина Тьюринга M *работает за время* $T(n)$, если максимальное по всем входам длиной n число шагов, которое сделает машина Тьюринга до остановки, равно $T(n)$. Время работы также называется *сложностью*.

Определение 24. Будем говорить, что алгоритм имеет *полиномиальную сложность* (работает за полиномиальное время), если его сложность равна $O(n^k)$ для некоторой постоянной k.

Определение 25. Назовем задачу *полиномиальной*, если для нее существует алгоритм, имеющий полиномиальную сложность.

Определение 26. Множество всех полиномиальных задач распознавания называется *классом задач* P.

Введем теперь важное понятие класса задач NP.

Определение 27. Язык L над алфавитом A принадлежит к классу *задач* NP в том и только в том случае, если произвольное $x \in A^*$ принадлежит языку L тогда и только тогда, когда существуют слово $s \in A^*$ и машина Тьюринга M, которая допускает пару (x, s) за полиномиальное время. Слово s будем называть сертификатом.

Другими словами, пусть существует злой волшебник Саруман, который мгновенно решает любую задачу. Мы, однако, ему не верим и просим, чтобы он нам доказал правильность своего решения. Не являясь волшебниками, мы можем решать за разумное время лишь полиномиальные задачи. Если для некоторой задачи распознавания Саруман, желающий убедить нас в своей правоте, может сообщить нам информацию, называемую сертификатом, пользуясь которой мы можем проверить его решение за полиномиальное время, то такая задача принадлежит к классу задач NP.

Очевидно, что $P \subseteq NP$. Однако до сих пор не известно, равны ли эти два класса.

Кроме классов сложности, связанных с временем работы машины Тьюринга, могут быть введены классы сложности, связанные с количеством ячеек ленты, которые машина Тьюринга использует в процессе работы.

Определение 28. Классом *PSPACE* называется множество задач распознавания, которые могут быть решены на машине Тьюринга, использующей $O(n^k)$ ячеек ленты, где n – длина входа, а k – некоторая постоянная.

Теорема 6. Имеет место включение $P \subseteq PSPACE$.

Доказательство. Очевидно, что число различных ячеек, посещенных головкой машины Тьюринга, не превышает числа совершенных ею шагов, которое для задач из класса P ограничено полиномом.

Теорема 7. Имеет место включение $NP \subseteq PSPACE$.

Доказательство. Задача проверки одной пары (x,s), где x – вход, а s – сертификат, согласно определению 27, принадлежит к классу P. Проверки различных пар независимы. Таким образом, можно последовательно проверять все сертификаты, для чего в соответствии с теоремой 6 надо посетить полиномиальное число ячеек, что и требовалось доказать.

Итак, о рассмотренных классах сложности известно, что

$$P \subseteq NP \subseteq PSPACE.$$

Определение 29. Задача $A(x)$ полиномиально сводится к задаче $B(x)$, если существует такая вычислимая за полиномиальное время функция f, что $A(x) = B(f(x))$.

Часто полиномиальную сводимость называют сводимостью по Карпу или просто сводимостью.

Пусть задача A сводится к задаче B. Тогда если задача A имеет сложность $c_A(n)$, а задача B – сложность $c_B(n)$, то выполняется неравенство $c_A(n) \le c_B(n) + O(n^k)$. Следовательно, если $B \in P$, то $A \in P$, а если $A \notin P$, то $B \notin P$.

4. *NP*-ПОЛНОТА

Существуют задачи, к любой из которых может быть сведена за полиномиальное время любая задача из класса *NP*. Такие задачи называются *NP*-полными. Эти задачи можно считать самыми

сложными в классе *NP*. В настоящее время известно более 3000 *NP*-полных задач. Хороший обзор *NP*-полных задач можно найти в работе [5]. Мы рассмотрим лишь некоторые из них.

Определение 30. Задачей о выполнимости называется следующая задача: дана булева формула над базисом $\{\vee, \wedge, \neg\}$. Определить, существует ли набор аргументов, на котором эта формула истинна.

Задача о выполнимости принадлежит к классу *NP*. В самом деле, имея в качестве сертификата набор, на котором формула истинна, можно за полиномиальное относительно длины формулы время проверить, действительно ли она является истинной, просто вычислив эту формулу.

Теорема 8 (*теорема Кука – Левина*[1]). Задача о выполнимости является *NP*-полной.

Доказательство. Строгое доказательство этой теоремы весьма громоздко и изобилует техническими деталями, поэтому мы представим лишь набросок доказательства.

Если число шагов, совершаемых машиной Тьюринга, полиномиально, то число ячеек ленты, которые она может использовать, также ограничено полиномом. Исходя из этого, весь процесс вычислений на входе *x* длиной *n* можно представить таблицей размера $u \times v$, где $u = O(n^k)$; $v = O(n^k)$ (табл. 3). Номер строки в этой таблице задает номер такта, а в каждой ячейке таблицы в двоичном виде закодирован элемент множества $A \times \{Q \cup \{\varnothing\}\}$, представляющий собой пару (символ в соответствующей ячейке ленты и состояние машины Тьюринга на данном такте работы в случае, если головка машины обозревает соответствующую ячейку ленты, в противном случае – символ $\varnothing$). Длина кода элемента множества $A \times \{Q \cup \{\varnothing\}\}$ не зависит от *n*. В верхней строке таблицы содержатся входное слово *x* и сертификат *s*, разделенные вспомогатель-

[1] Эта основополагающая теорема была впервые доказана советским математиком Л. Левиным в 1971 г. В том же году независимо от него она была доказана американским математиком С. Куком. Поскольку статья Л. Левина была опубликована лишь в 1973 г., долгое время эта теорема называлась теоремой Кука. Но тот факт, что Л. Левин делал доклады о своих результатах на научных конференциях в 1971 г., позволил установить приоритет Л. Левина.

ным символом «#». В последней строке таблицы в одной из ячеек записано состояние, в котором произведен останов.

Таблица 3

Таблица к доказательству теоремы Кука – Левина

0	…	x_1, q_0	$x_2, \varnothing$	$x_3, \varnothing$	…	$x_n, \varnothing$	$\#, \varnothing$	$s_1, \varnothing$	…	$s_m, \varnothing$	…
1											
	…										
$i-1$		$\lambda_{i-1,j-1}$	$\lambda_{i-1,j}$	$\lambda_{i-1,j+1}$		…					
i		…	$\lambda_{i,j}$	…							
…	…										
u											

Введем булеву функцию ξ, зависящую от входа, которая истинна тогда и только тогда, когда машина Тьюринга остановилась в допускающем состоянии. Заметим, что эту функцию можно рассматривать как зависящую лишь от переменных, входящих в последнюю строку таблицы.

Легко увидеть, что содержимое *j*-й ячейки *i*-й строки таблицы $(i \neq 0)$ зависит не более чем от трех ячеек с номерами $j-1, j$ и $j+1$ строки $i-1$. Таким образом, вычислить значение каждой ячейки можно с помощью набора булевых функций. То есть для каждой четверки ячеек такого вида должны выполняться правила согласования, которые можно представить в виде набора булевых формул. Определим булеву формулу ψ_x как конъюнкцию всех этих формул и формулы функции ξ. Формула ψ_x зависит только от сертификата *s*. Сложность ее построения полиномиальна, так как пропорциональна количеству ячеек таблицы, равному $O(n^{2k})$. Согласно построению эта формула выполнима тогда и только тогда, когда существует такой сертификат, что машина Тьюринга останавливается в допускающем состоянии. Из этого факта следует утверждение теоремы. Строгое доказательство этой теоремы можно найти, например, в работе [12].

Определение 31. Задачей о 3-выполнимости (в англоязычной литературе: 3-*SAT*) называется следующая задача: дана конъюнктивная нормальная форма (КНФ), каждая дизъюнкция которой содержит ровно три переменные (такую формулу будем называть 3-КНФ). Определить, существует ли набор аргументов, на котором 3-КНФ истинна.

Теорема 9. Задача о 3-выполнимости является *NP*-полной.

Доказательство. Задача о 3-выполнимости принадлежит к классу *NP*. Действительно, в качестве сертификата можно взять набор аргументов, на котором 3-КНФ истинна.

Покажем, что задачу о выполнимости можно свести к задаче о 3-выполнимости. Пусть $\varphi(x_1,...,x_m)$ – формула над базисом $\{\vee,\wedge,\neg\}$. Нам надо за полиномиальное время получить 3-КНФ ψ, которая была бы выполнимой тогда и только тогда, когда выполнима формула φ. С этой целью мы рассмотрим дерево разбора формулы φ. Листья этого дерева будут соответствовать литералам (т. е. переменным либо их отрицаниям), а узлы дерева – операциям из базиса $\{\vee,\wedge,\neg\}$.

$$\overline{a \vee b} = \overline{a} \wedge \overline{b};$$

$$\overline{a \wedge b} = \overline{a} \vee \overline{b};$$

$$\overline{\overline{a}} = a.$$

Введем дополнительные переменные $y_1 \ldots y_r$, соответствующие узлам дерева. Для каждой из этих переменных истинна одна из следующих формул:

$$y_i \sim (z_i \vee t_i); \tag{4}$$

$$y_i \sim (z_i \wedge t_i), \tag{5}$$

где z_i и t_i – соответственно левый и правый потомки каждого узла дерева, представляющие собой дополнительные переменные y или литералы.

Обозначим символом ϑ_i формулу (4), если она истинна, или формулу (5), если истинна она.

Теперь запишем

$$\omega(x_1 \dots x_m,\ y_1 \dots y_r) = \bigwedge_{i=1}^{r} \vartheta_i.$$

Отметим, что формула ω является выполнимой тогда и только тогда, когда выполнима формула φ. Действительно, если существует набор, на котором истинна формула φ, дополнительные переменные формулы ω можно получить, исходя из того, что для каждого узла дерева истинно выражение (3) или (4) в зависимости от соответствующей узлу операции. Заметим, что если на некотором наборе истинна формула ω, то, взяв из него только переменные *x*, мы обнаружим, что на этом наборе истинна формула φ.

Теперь преобразуем формулы ϑ_i, соответствующие каждому узлу дерева в 3-КНФ. Для каждой формулы это можно сделать, построив таблицу истинности и записав по ней КНФ обычным способом. Сложность такого преобразования для каждой формулы не зависит от длины входа. Обозначим такие КНФ как μ_i. Остается записать формулу

$$\psi(x_1 \dots x_m,\ y_1 \dots y_r) = \bigwedge_{i=1}^{r} \mu_i, \qquad (6)$$

которая и является искомой 3-КНФ.

Итак, у нас теперь есть *NP*-полная задача и мы можем доказывать *NP*-полноту других задач с помощью сводимости. Покажем этот метод на примере задачи о клике.

Определение 32. Кликой размера *k* в графе *G* называется полный подграф графа *G*, состоящий из *k* вершин.

Определение 33. Задача о клике формулируется следующим образом: даны неориентированный граф *G* и натуральное число *k*. Требуется распознать, существует ли в графе *G* клика размером *k*.

Задачу о клике можно решить путем полного перебора всех возможных комбинаций из *k* вершин и проверки, не образует ли каждая из этих комбинаций клику. Очевидно, что сложность этого равна

$$L_{clique} = \Omega\left(k^2 C_n^k\right), \qquad (7)$$

где *n* – число вершин в графе.

В худшем случае

$$k = \frac{n}{2}$$

и

$$L_{clique} = \Omega(n^2 C_n^{n/2}) = \Omega(n \cdot 2^n) .$$

Как мы уже упоминали, граф можно закодировать в конечном алфавите. Существует целый ряд различных способов кодирования графов. Наиболее известными являются представления в виде матрицы смежности, матрицы инцидентности и списка вершин. Подробнее об этих способах можно прочитать в работах [6, 9].

Теорема 10. Задача о клике принадлежит к классу *NP*.

Доказательство. Действительно, в качестве сертификата мы можем взять список вершин, образующих клику. Наличие всех ребер графа, соединяющих эти вершины, можно проверить за полиномиальное время.

Теорема 11. Задача о клике является *NP*-полной.

Доказательство. Сведем задачу о 3-выполнимости к задаче о клике. Используя сводящий алгоритм, получим 3-КНФ, выполнимость которой нужно проверить. Пусть дана формула

$$\psi = \bigwedge_{j=1}^{k} c_j ,$$

где каждая c_j есть дизъюнкция ровно трех литералов.

Построим граф *G*, который содержит клику размера *k* в том и только в том случае, если формула ψ выполнима. С этой целью каждой дизъюнкции c_j сопоставим тройку вершин графа *G* – по одной вершине для каждого литерала.

Опишем ребра этого графа. Две вершины соединены ребром в том и только в том случае, когда одновременно выполняются следующие два условия:

- литералы вершин принадлежат разным дизъюнкциям;
- литералы вершин не являются отрицанием друг друга.

Отметим, что если формула ψ истинна на некотором наборе, то на этом наборе каждая дизъюнкция c_j имеет не менее одного истинного литерала. Выберем по одному истинному литералу из

каждой дизъюнкции. Эти литералы входят в разные дизъюнкции и при этом не являются отрицанием друг друга (так как они все истинны). Таким образом, вершины, соответствующие им, образуют клику размера k.

В то же время если в графе G есть клика размера k, то в нее входит ровно по одной вершине из каждой тройки (поскольку вершины из одной тройки не соединены). Если теперь литерал каждой вершины такой клики приравнять к единице, мы найдем набор, на котором формула ψ истинна.

Таким образом, если в графе G существует клика размером k, то существует и набор, на котором формула ψ истинна, и, наоборот, если существует набор, на котором формула ψ истинна, то существует и клика размера k. Очевидно, что граф G по формуле ψ можно построить за полиномиальное время.

Итак, мы доказали, что задача о 3-выполнимости сводится к задаче о клике. Из этого следует, что задача о клике является *NP*-полной.

Докажем теперь *NP*-полноту еще одной задачи – задачи о вершинном покрытии.

Определение 34. Задача о вершинном покрытии формулируется следующим образом: даны граф $G = (V, E)$ и число k. Требуется распознать, существует ли в графе G вершинное покрытие размера k, т. е. множество из k вершин, такое, что любое ребро графа было бы инцидентно хотя бы одной вершине из этого множества.

Теорема 12. Задача о вершинном покрытии является *NP*-полной.

Доказательство. Очевидно, что задача о вершинном покрытии принадлежит к классу *NP*: в качестве сертификата достаточно взять вершинное покрытие.

Теперь докажем собственно *NP*-полноту. Для этого сведем задачу о клике к задаче о вершинном покрытии. Рассмотрим дополнение к графу G, т. е. граф $\overline{G} = (V, \overline{E})$, где $\overline{E} = \{(u,v)\colon (u,v) \notin E\}$. В графе $\overline{G}$ те же вершины, что и в графе G, но ребра соединяют те и только те пары вершин, которые не соединены ребрами в графе G. Заметим, что граф $\overline{G}$ имеет вершинное покрытие размером $n-k$ (где n – число вершин в графе) в том и только в том случае, когда граф G имеет клику размером k. Действительно, все вершины, не входящие в клику размером k графа G, образуют вершинное по-

крытие размером $n-k$ в графе $\overline{G}$, так как в графе $\overline{G}$ имеются те и только те ребра, которых нет в графе G, т. е. нет ребер, которые соединяли бы две вершины, входящие в клику. В то же время все вершины, не входящие в вершинное покрытие графа $\overline{G}$, являются кликой в графе G, поскольку если какие-то две такие вершины не являются смежными в графе G, то они являются смежными в графе $\overline{G}$, чего быть не может, так как они не входят в вершинное покрытие.

Таким образом, задача о клике может быть решена путем решения задачи о вершинном покрытии с использованием дополнения к графу. И наоборот, задача о вершинном покрытии может быть решена путем решения задачи о клике в дополнении к графу. При этом дополнение к графу строится за полиномиальное время. Из этого следует утверждение теоремы.

С классом *NP* связана одна из самых известных открытых проблем современной математики – доказательство гипотезы о том, что $P \neq NP$. Говоря неформально, эта гипотеза означает, что существуют задачи, решить которые гораздо сложнее, чем проверить готовое решение.

5. СХЕМЫ ИЗ ФУНКЦИОНАЛЬНЫХ ЭЛЕМЕНТОВ

Перейдем теперь к другому способу исследования сложности алгоритмов – к так называемым схемам из функциональных элементов.

Пусть B – множество, состоящее из не более чем двухместных булевых функций.

Определение 35. *Схемой из функциональных элементов* с n входами и m выходами над базисом B назовем ориентированный ациклический граф, обладающий следующими свойствами:

- граф содержит n вершин с входной степенью, равной нулю. Такие вершины называются входами. Входы пронумерованы от 1 до n;
- остальные вершины графа называются элементами схемы и имеют входные степени, не превышающие двух;
- с i-м входом схемы ассоциирована функция $f(x_1,...,x_n) = x_i$;

- ребра, входящие в элемент, нумеруются числами от 1 до 2;
- с каждым элементом схемы ассоциирована функция из B, причем количество переменных этой функции равно входной степени элемента;
- некоторые m вершин пронумерованы числами $1 \ldots m$. Эти вершины называются выходами схемы.

Определение 36. Если в схеме из функциональных элементов вершины v и u связаны ребром, ориентированным от u к v и имеющим номер i, то говорят, что вершина v присоединена к вершине u. При этом u называется i-м *предком* вершины v, а v – *потомком* вершины u.

Определение 37. Определим индуктивно функцию $S_w(x_1, ..., x_n)$, вычисляемую вершиной w; f_w – функция, ассоциированная с этой вершиной. Тогда, если у вершины w имеются t предков $(u_1, ..., u_t)$, будем говорить, что эта вершина вычисляет функцию $S_w = f(S(u_1), ..., S(u_t))$. Схема вычисляет набор функций, вычисляемых выходами схемы.

Схемы из функциональных элементов являются способом вычисления булевых функций. Схеме можно сопоставить две важные характеристики: сложность и глубину.

Определение 38. *Сложностью* $L(S)$ схемы S называется количество элементов схемы.

Определение 39. *Глубиной* $D(S)$ схемы S называется число элементов максимальной ориентированной цепи среди всех цепей, соединяющих входы и выходы схемы.

По схеме из функциональных элементов можно построить так называемую неветвящуюся программу.

Определение 40. *Неветвящейся программой* над базисом B и множеством независимых переменных $\{x_1, \ldots, x_n\}$ называется конечная последовательность равенств, каждое из которых имеет один из трех видов:

- $y_i = f_i(u_i, v_i)$, если f_i – функция от двух переменных;
- $y_i = f_i(u_i)$, если f_i – функция от одной переменной;
- $y_i = f_i$, если f_i – константа,

где $f_i \in B$; $i = 1, ..., L$; $u_i, v_i \in \{x_1, \ldots, x_n, y_1, \ldots, y_{i-1}\}$. Такие равенства назовем *командами*.

Переменные y_i назовем *внутренними переменными*. Некоторые переменные неветвящейся программы назовем *выходными*. Для удобства неветвящуюся программу будем предварять ее именем с указанием параметров, а для указания выходных переменных в конце будем записывать слово return со списком выходных переменных.

Пример неветвящейся программы, которая реализует одноразрядный сумматор, получающий на входе два операнда и перенос и выдающий на выходе сумму и перенос в старший разряд:

$$\begin{aligned}
&\mathrm{sum}(x,\ y,\ c):\\
&t_1 = x \oplus y\\
&s = t_1 \oplus c\\
&t_2 = t_1 \cdot c\\
&t_3 = x \cdot y\\
&c' = t_2 \oplus t_3\\
&\mathrm{return}(s,\ c')
\end{aligned}$$

Неветвящейся программе можно поставить в соответствие схему, просто строя вершину для каждой команды по порядку следования в программе. По схеме можно построить неветвящуюся программу, упорядочив все узлы по глубине и поставив в соответствие каждому узлу переменную и команду.

Определение 41. Назовем *сложностью* неветвящейся программы число команд в ней.

Очевидно, что сложность программы равна сложности соответствующей схемы.

Определение 42. *Глубиной* неветвящейся программы будем называть глубину соответствующей ей схемы.

Фактически схема из функциональных элементов и неветвящаяся программа являются различными способами записи одного и того же объекта.

Любая булева функция может быть задана бесконечным количеством различных схем. Интерес представляет схема, имеющая минимальную сложность.

Определение 43. Сложностью $L(f)$ булевой функции f называется минимальная сложность схемы, вычисляющей эту функцию.

Перейдем теперь к оценкам схемной сложности. Если это не указывается особо, используем стандартный базис $\{\vee,\ \wedge,\ \neg\}$.

Выберем среди всех булевых функций от n переменных функцию с наибольшей сложностью и обозначим эту сложность L_n. Оценим эту величину сверху.

Теорема 13. Имеет место неравенство $L_n \le 3 \cdot 2^n - 4$.

Доказательство. Заметим сначала, что $L_1 = 2$ (такой сложностью обладают константы 0 и 1). Предположим теперь, что утверждение теоремы справедливо для $n = k$. Покажем, что оно выполняется для $n = k+1$. Действительно, пусть f – наиболее сложная функция от n переменных. Используем разложение функции по последней переменной:

$$f(x_1,..., x_k, x_{k+1}) = x_{k+1} f(x_1,..., x_k, 1) \vee \overline{x}_{k+1} f(x_1,..., x_k, 0). \qquad (8)$$

Тогда если функция $f(x_1, ..., x_k, 1)$ вычисляется программой S_1, а $f(x_1,..., x_k, 0)$ – программой S_0, то $f(x_1,..., x_k, x_{k+1})$ вычисляется следующей программой:

$S(x_1,..., x_{k+1})$:
$y_0 = S_0(x_1,..., x_k, 0)$
$y_1 = S_1(x_1,..., x_k, 1)$
$y_2 = \overline{x}_{k+1}$
$y_3 = y_0 \cdot y_2$
$y_4 = y_1 \cdot x_{k+1}$
$z = y_3 \vee y_4$
return(z)

Таким образом,

$$L_{k+1} \le L(S_0) + L(S_1) + 4 =$$

$$= 2L_k + 4 \le 2(3 \cdot 2^k - 4) + 4 = 3 \cdot 2^{k+1} - 4, \qquad (9)$$

что и требовалось доказать.

Теорема 14. Существует неветвящаяся программа K_k, вычисляющая все элементарные конъюнкции от k переменных и имеющая сложность $L(K_k) \le 2^{k+1} + k$.

Доказательство. Программа K_1 имеет вид

$$K_1(x):$$
$$y = \overline{x}$$
$$\text{return}(x, y)$$

Сложность этой программы $L(K_1) = 1$.

Программа K_k записывается следующим образом:

$$K_k(x_1, ..., x_k):$$
$$(y_1, ..., y_{2^{k-1}}) = K_{k-1}(x_1, ..., x_{k-1})$$
$$t = \overline{x}_k$$
$$z_1 = y_1 \cdot x_k$$
$$z_2 = y_2 \cdot x_k$$
$$\vdots$$
$$z_{2^{k-1}} = y_{2^{k-1}} \cdot x_k$$
$$z_{2^{k-1}+1} = y_1 \cdot t$$
$$z_{2^{k-1}+2} = y_2 \cdot t$$
$$\vdots$$
$$z_{2^k} = y_{2^{k-1}} \cdot t$$
$$\text{return}\left(z_1, ..., z_{2^k}\right)$$

Таким образом,

$$L(K_k) = L(K_{k-1}) + 2^k + 1 = 2^k + 2^{k-1} + ... + 2^2 + k = 2^{k+1} + k - 4,$$

что и требовалось доказать.

Теорема 15 (*теорема Шеннона*). Для сложности функции $f(x_1, ..., x_n)$ справедливо соотношение $L(f) \le 8 \cdot \frac{2^n}{n} \cdot \left(1 + o(1)\right)$.

Доказательство. Разложим функцию $f(x_1, ..., x_n)$ по первым k переменным:

$$f(x_1, ..., x_n) = \bigvee_{\sigma_1, ..., \sigma_k} f(\sigma_1, ..., \sigma_k, x_{k+1}, ..., x_n) \cdot x_1^{\sigma_1}, \ldots, x_k^{\sigma_k}. \quad (10)$$

Итак, нам надо вычислить функцию f. В соответствии с формулой (10) и доказанными выше теоремами 13 и 14 для этого требуется вычислить:

1) все элементарные конъюнкции от k переменных, сложность чего не превышает $L(K_k) \le 2^{k+1} + k$;

2) всевозможные функции от $n-k$ переменных. Таких функций $2^{2^{n-k}}$. Следовательно, сделать это можно со сложностью, не превышающей $3 \cdot 2^{n-k} \cdot 2^{2^{n-k}}$;

3) еще 2^k и $2^k - 1$ дизъюнкции.

Таким образом, для общей сложности имеем неравенство $L(f) \le 2^{k+1} + k + 3 \cdot 2^{n-k} \cdot 2^{2^{n-k}} + 2 \cdot 2^k = 4 \cdot 2^k + O(2^{n-k} \cdot 2^{2^{n-k}})$.

Положив $k = \lceil n - \log_2(n - 3\log_2 n) \rceil$, получим

$$L(f) \le 8 \cdot \frac{2^n}{n} + O(n \cdot \frac{2^n}{n^3}) = 8 \cdot \frac{2^n}{n} \cdot \big(1 + o(1)\big).$$

Теорема доказана.

Мы получили оценку сверху для сложности произвольной булевой функции. Эту оценку, однако, можно улучшить.

Теорема 16 (*теорема Лупанова*). Для сложности функции $f(x_1, ..., x_n)$ справедливо соотношение $L(f) \le \frac{2^n}{n} \cdot \big(1 + o(1)\big)$.

Доказательство этой теоремы выходит за рамки учебного пособия. Заинтересованный читатель найдет его в работах [10, 15].

Теорема 17. Доля булевых функций от n переменных, для сложности которых выполняется неравенство

$$L(f) \ge \frac{2^n}{n},$$

стремится к единице при стремящемся к бесконечности n.

Доказательство. Оценим сверху число $S_0(n,b,h)$ минимальных программ над базисом мощности b, имеющих одну выходную переменную, n входных переменных и ровно h команд. Заметим, что одну команду мы можем выбрать не более чем $b(n+h)^2$ спо-

собами, следовательно, h команд можно выбрать $(b(n+h)^2)^h$ способами. Учтем теперь, что при таком методе подсчета для каждой оптимальной программы мы учитываем все ее экземпляры, различающиеся нумерацией внутренних переменных. Легко увидеть, что число таких экземпляров равно числу перестановок внутренних переменных и составляет $h!$. В результате получим следующее неравенство:

$$S_0(n, b, h) \le \frac{(b(n+h)^2)^h}{h!}.$$

Оценим теперь число $S(n, b, h)$ минимальных программ, состоящих не более чем из h команд:

$$S(n, b, h) = \sum_{k=0}^{h} S_0(n, b, k) \le (h+1)\frac{(b(n+h)^2)^h}{h!} \le \frac{b^h(n+h)^{2h+1}}{h!}.$$

Считая, что $h > n$ и используя формулу Стирлинга

$$n! \sim \sqrt{2\pi n}\left(\frac{n}{e}\right)^n,$$

получаем

$$S(n, b, h) \le \frac{b^h(n+h)^{2h+1}}{\left(\frac{h}{e}\right)^h} \le (\mu h)^h,$$

где μ – некоторая константа.

Положив $h = \left\lfloor \frac{2^n}{n} \right\rfloor$, имеем

$$S\left(n, b, \left\lfloor \frac{2^n}{n} \right\rfloor\right) \le \left(\mu \frac{2^n}{n}\right)^{\frac{2^n}{n}}.$$

Рассмотрим теперь выражение

$$\lim_{n\to\infty}\left(\log_2\frac{S\left(n,b,\left\lfloor\frac{2^n}{n}\right\rfloor\right)}{2^{2^n}}\right)\le\lim_{n\to\infty}\left(\log_2\frac{\left(\mu\frac{2^n}{n}\right)^{\frac{2^n}{n}}}{2^{2^n}}\right)=$$

$$=\lim_{n\to\infty}\left(\frac{2^n}{n}\left(\log_2\mu+n-\log_2 n\right)-2^n\right)=\lim_{n\to\infty}\left(\frac{2^n}{n}\left(\log_2\mu-\log_2 n\right)\right)=-\infty.$$

Следовательно, с ростом числа аргументов доля функций, сложность которых $L(f)\ge\frac{2^n}{n}$, асимптотически стремится к единице, что и требовалось доказать.

Таким образом, из двух предыдущих теорем 15 и 16 можно сделать вывод о том, что почти все булевы функции имеют очень большую сложность, а именно $\frac{2^n}{n}$, тем не менее указать конкретную последовательность функций экспоненциальной сложности до сих пор не удается.

Теперь перейдем к исследованию связи неветвящихся программ и языков.

Определение 44. Назовем *семейством* неветвящихся программ последовательность C_n, $n\in\mathbb{N}$, неветвящихся программ, имеющих n входных переменных.

Определение 45. Будем говорить, что семейство неветвящихся программ C_n распознает язык L над алфавитом $\{0;1\}$, если слово x длиной n принадлежит языку L тогда и только тогда, когда $C_n(x)=1$.

Определим теперь класс языков *P/poly*, связанный с понятием схемной сложности.

Определение 46. Язык U над алфавитом $\{0;1\}$ принадлежит к классу *P/poly*, если существует семейство неветвящихся программ C_n, распознающее язык U, причем сложность $L(C_n)$ полиномиально зависит от n.

Теорема 18. Имеет место включение $P \subseteq P/poly$.

Доказательство. Для доказательства теоремы достаточно показать, что любая машина Тьюринга, работающая за полиномиальное время, может быть представлена в виде последовательности булевых функций, имеющей полиномиальную сложность. Доказательство этого аналогично доказательству теоремы 8 (теоремы Кука – Левина).

Теорема 19. Любой *унарный* язык (т. е. язык $L \subseteq \left\{1^n \mid n \in \mathbb{N}\right\}$ над алфавитом $\{0; 1\}$) принадлежит к классу $P/poly$.

Доказательство. Рассмотрим произвольный унарный язык L. Опишем последовательность неветвящихся программ, допускающих этот язык:

- если $1^n \in L$, то соответствующая неветвящаяся программа состоит из $n-1$ конъюнкций;
- если $1^n \notin L$, то соответствующая неветвящаяся программа реализует тождественный нуль.

Таким образом, имеется последовательность неветвящихся программ, допускающих произвольный унарный язык, сложность которых линейно зависит от числа переменных. Следовательно, любой унарный язык принадлежит к классу $P/poly$, что и требовалось доказать.

Теорема 20. Имеет место неравенство $P \neq P/poly$.

Доказательство. Возьмем язык L_s, состоящий из кодов самоприменимых машин Тьюринга. Рассмотрим язык L, представляющий собой унарный вариант языка L_s:

$$L = \left\{1^j,\ j \in L_s\right\}.$$

Очевидно, что язык L алгоритмически неразрешим и, следовательно, не принадлежит к классу P. Однако язык L является унарным и согласно теореме 19 принадлежит к классу $P/poly$. Теорема доказана.

Итак, мы видим, что

$$P \subset P/poly.$$

Показав, что в классе $P/poly$ содержатся некоторые алгоритмически неразрешимые языки, мы увидели, что класс $P/poly$ шире класса P. Однако до сих пор не известно, является ли класс NP подмножеством класса $P/poly$ или нет.

6. ВЕРОЯТНОСТНЫЕ АЛГОРИТМЫ

До сих пор в качестве алгоритмической модели мы рассматривали машину Тьюринга. У этой модели есть один недостаток: она не может делать случайный выбор во время работы. Алгоритмы, использующие в своей работе случайные числа, называются вероятностными. В качестве алгоритмической модели, описывающей такие алгоритмы, выступает вероятностная машина Тьюринга.

Определение 47. *Вероятностной машиной Тьюринга* называется машина Тьюринга, имеющая две функции перехода: δ_0 и δ_1. На каждом шаге с вероятностью $1/2$ применяется функция δ_0 и с вероятностью $1/2$ – функция δ_1.

Вероятностные алгоритмы бывают двух типов. Первый тип получает правильный ответ лишь с некоторой вероятностью, второй тип – всегда, несмотря на то что алгоритм использует случайные числа. В связи с этим существуют два вероятностных аналога класса *P*: класс *BPP* и класс *ZPP*.

Определение 48. *Я*зык L принадлежит к классу *BPP*, если существует вероятностная машина Тьюринга M, работающая за полиномиальное время, которая:

- допускает слово x с вероятностью, большей $2/3$, для любого $x \in L$;
- не допускает слово x с вероятностью, большей $2/3$, для любого $x \notin L$.

Определение 49. *Я*зык L принадлежит к классу *ZPP*, если существует вероятностная машина Тьюринга, работающая за полиномиальное время и допускающая этот язык.

Заметим, что вероятность, фигурирующую в определении класса *BPP*, можно сделать сколь угодно близкой к единице. Действительно, запустим вероятностную машину Тьюринга k раз на одном и том же входе x и будем считать ответом результат, который получился большее число раз. Тогда в соответствии с известным из теории вероятностей неравенством Чернова [11] вероятность того, что результат правильный, составляет

$$P \geq 1 - \mathrm{e}^{-2k\left(p-\frac{1}{2}\right)^2},$$

где вероятность каждого события p согласно определению вероятностной машины Тьюринга равна $2/3$. Таким образом,

$$P \geq 1 - \mathrm{e}^{-\frac{1}{18}k}.$$

Следовательно, увеличивая k, мы можем получить величину P, сколь угодно близкую к единице.

В качестве примера приведем вероятностный алгоритм проверки чисел на простоту. Для этого используем малую теорему Ферма, известную из теории чисел.

Теорема 21 (*малая теорема Ферма*). Если p – простое число, то для любого натурального числа a выполняется сравнение $a^p = a \pmod{p}$.

Доказательство. Очевидно, что для $a = 1$ теорема верна. Пусть выполняется сравнение $(a-1)^p = a - 1 \pmod{p}$. Тогда, учитывая, что при $0 < k < p$ число C_p^k делится нацело на p, получаем

$$a^p = \big((a-1)+1\big)^p = \sum_{k=0}^{p} C_p^k (a-1)^k = (a-1)^p + 1 =$$
$$= a - 1 + 1 = a \pmod{p},$$

что и требовалось доказать.

Теорема 22. Если n – составное число и существует $a \in \{1, 2, ..., n-1\}$, такое, что $a^{n-1} \neq 1 \pmod{n}$, то не менее чем для половины различных $b \in \{1, 2, ..., n-1\}$ выполняется неравенство $b^{n-1} \neq 1 \pmod{n}$.

Доказательство. Пусть $\{b_1, b_2, \ldots, b_s\}$ – множество всех оснований, для которых $b_k^{n-1} = 1 \pmod{n}$. Рассмотрим произведения вида ab_k, для которых

$$(ab_k)^{n-1} = a^{n-1} \cdot b_k^{n-1} = a^{n-1} \neq 1 \pmod{n}.$$

Отсюда, поскольку таких произведений ab_k столько же, сколько и b_k, следует утверждение теоремы.

Согласно теоремам 21 и 22 число n можно проверить на простоту с помощью следующего алгоритма:

1) выбрать случайное основание $a \in \{1, 2, ..., n-1\}$;

2) вычислить $z = a^{n-1} \pmod n$.

Если $z \neq 1$, то число n точно составное. Если $z = 1$, вероятность того, что n – простое число, не меньше $1/2$.

Если этот алгоритм выполнить k раз и каждый раз окажется, что $z = 1$, то вероятность того, что n – число простое, окажется не меньшей чем $1 - 2^{-k}$.

Изложенный алгоритм редко используется на практике ввиду наличия чисел, являющихся составными, но для которых выполняется сравнение $a^{n-1} = 1 \pmod n$ для любых a. Такие числа называются числами Кармайкла, и их бесконечно много. Примером такого числа является число 561. Встречаются числа Кармайкла не очень часто. Так, среди первого миллиарда натуральных чисел их всего 646. К счастью, есть вероятностные тесты на простоту, свободные от этого недостатка, например алгоритмы Рабина – Миллера и Соловея – Штрассена. Их рассмотрение выходит за рамки данного учебного пособия, но, учитывая важность этих алгоритмов для ряда приложений, в том числе для криптографии, авторы рекомендуют ознакомиться с их подробным описанием в работах [8, 13].

Итак, мы привели пример (правда, не совсем совершенный) вероятностного алгоритма.

Очевидно, что классы *BPP* и *ZPP* включают в себя класс *P*. Но возникают вопросы: равны ли эти классы классу *P*, или они шире него? равны ли классы *ZPP* и *BPP* между собой, или нет? Пока не известны ответы на эти вопросы.

7. ЗАЩИТА ИНФОРМАЦИИ И ТЕОРИЯ АЛГОРИТМОВ

Теория алгоритмов очень важна для различных аспектов информационной безопасности. В частности, все асимметричные и практически все симметричные криптоалгоритмы основаны на однонаправленных функциях.

Определение 50. Однонаправленной функцией называется функция $f : \{0; 1\}^* \to \{0; 1\}^*$, такая, что функция $y = f(x)$ может

быть вычислена за полиномиальное время, в то время как обратить ее, т. е. вычислить x, имея y, можно только за неполиномиальное время.

В настоящее время не доказано существование однонаправленных функций. Однако существуют примеры функций, вычислимых за полиномиальное время, но для которых не известен полиномиальный алгоритм обращения. Такие функции и предполагаются однонаправленными.

Очевидно, что если однонаправленные функции существуют, то $P \neq NP$. В то же время из неравенства $P \neq NP$ не следует существования однонаправленных функций.

Примером функции, которая предполагается однонаправленой, может служить умножение целых чисел. Имея в качестве входа два числа, можно за полиномиальное время (например, с помощью алгоритма умножения в столбик) вычислить их произведение. В то же время осуществить факторизацию, т. е. по произведению найти сомножители, в общем случае трудно. Лучший известный алгоритм факторизации (алгоритм решета числового поля [2]) имеет сложность $e^{(c+o(1))\sqrt[3]{(\log n)(\log\log n)^2}}$. На однонаправленности умножения целых чисел (другими словами, на неполиномиальности задачи факторизации) основана стойкость криптосистемы RSA.

Криптографические хэш-функции и блочные шифры (если рассматривать их функции шифрования как функции от ключа) также предположительно являются однонаправленными.

Стойкость всех современных криптографических алгоритмов, длина ключа которых меньше длины открытого текста, основана на гипотезах о существовании однонаправленных функций и о том, что $P \neq NP$.

ЗАДАЧИ ДЛЯ САМОСТОЯТЕЛЬНОГО РЕШЕНИЯ

Задачи оценены в баллах в зависимости от их сложности. Минимальная сложность соответствует одному баллу.

Задача 1. Построить машину Тьюринга, вычитающую единицу из двоичного числа. (1 балл)

Задача 2. Построить машину Тьюринга, складывающую два двоичных числа. (1 балл)

Задача 3. Построить инъективное отображение $f : T_m \to \mathbb{N}$, где T_m – множество всех машин Тьюринга над алфавитом мощности m. (2 балла)

Задача 4. Завод по производству машин Тьюринга выпустил партию бракованных машин Тьюринга. Они отличаются от обычных тем, что на них невозможно использовать направление движения головки *S*. Существуют ли задачи, разрешимые на обычной машине Тьюринга, но не разрешимые на бракованной машине Тьюринга? (3 балла)

Задача 5. В связи с перебоями в поставках бумажной ленты директором завода по производству машин Тьюринга принято решение промышленно выпускать машины Тьюринга, имеющие ленту, бесконечную только в одну сторону. Директор утверждает, что не существует задач, которые были бы разрешимы на обычной машине Тьюринга, но не разрешимы на машине Тьюринга с односторонней лентой. Прав ли директор? (3 балла)

Задача 6. Назовем музыкальной машиной Тьюринга машину Тьюринга, каждое состояние которой помечено одной из семи нот. Как только такая машина Тьюринга переходит в новое состояние, она играет соответствующую ему ноту. Доказать неразрешимость следующей задачи: сыграет ли данная музыкальная машина Тьюринга на данном входе ноту *до*? (2 балла)

Задача 7. Показать, что класс *P* замкнут относительно операций дополнения и объединения. (2 балла)

Задача 8. Доказать, что если $P = NP$, то задача факторизации целых чисел может быть решена за полиномиальное время. (2 балла)

Задача 9. Построить полиномиальный алгоритм для задачи о выполнимости 2-КНФ, т. е. КНФ, каждая дизъюнкция которой содержит ровно две переменные. (4 балла)

Задача 10. Доказать *NP*-полноту задачи целочисленного линейного программирования. Задача целочисленного линейного программирования формулируется следующим образом: даны целочисленные матрица A и векторы **b** и **c**. Найти такой целочисленный вектор **x**, чтобы он максимизировал скалярное произведение векторов $(\mathbf{c}, \mathbf{x})$ при выполнении условий $Ax \le b$. Указание: свести к этой задаче задачу о 3-выполнимости. (4 балла)

Задача 11. Доказать, что схемная сложность функции $f(x_1, \ldots, x_n) = x_1 \vee x_2 \vee \ldots \vee x_n$ над базисом $\{\neg, \wedge\}$ равна $2n$. (2 балла)

ЛИТЕРАТУРА

1. *Булос Д.*, *Джеффри Р.* Вычислимость и логика: Пер. с англ. М.: Мир, 1994. 396 с.
2. *Василенко О.Н.* Теоретико-числовые алгоритмы в криптографии. М.: Изд-во МЦНМО, 2003. 325 с.
3. *Верещагин Н.К.*, *Шень А.* Лекции по математической логике и теории алгоритмов: В 3 ч. Ч. 3. Вычислимые функции. М.: Изд-во МЦНМО, 2002. 299 с.
4. *Верещагин Н.К.*, *Шень А.* Лекции по математической логике и теории алгоритмов: В 3 ч. Ч. 2. Языки и исчисления. М.: Изд-во МЦНМО, 2002. 288 с.
5. *Гэри М.*, *Джонсон Д.* Вычислительные машины и трудно-решаемые задачи: Пер. с англ. М.: Мир, 1982. 416 с.
6. *Емеличев В.А.* и др. Лекции по теории графов. М.: URSS, 2009. 382 с.
7. *Китаев А.*, *Шень А.*, *Вялый М.* Классические и квантовые вычисления. М.: Изд-во МЦНМО, 1999. 192 с.
8. *Коблиц Н.* Курс теории чисел и криптографии. М.: ТВП, 2001. 260 с.
9. *Кормен Т.*, *Лейзерсон Ч.*, *Ривест Р.* Алгоритмы. Построение и анализ: Пер. с англ. М.: Вильямс, 2005. 1290 с.
10. *Лупанов О.Б.* Асимптотические оценки сложности управляющих систем: Учеб. пособие. М.: Изд-во Моск. гос. ун-та, 1984. 137 с.
11. *Феллер В.* Введение в теорию вероятностей и ее приложения: В 2 т. М.: Либроком, 2010. Т. 1. 511 с.; Т. 2. 766 с.
12. *Хопкрофт Д.Э.*, *Мотвани Р.*, *Ульман Д.* Введение в теорию автоматов, языков и вычислений: Пер. с англ. М.: Вильямс, 2008. 527 с.
13. *Черемушкин А.В.* Лекции по арифметическим алгоритмам в криптографии. М.: Изд-во МЦНМО, 2002. 103 с.
14. *Шоломов Л.А.* Основы теории дискретных логических и вычислительных устройств. М.: Наука, 1980. 399 с.
15. *Яблонский С.В.* Введение в дискретную математику. М.: Высш. шк., 2010. 384 с.
16. *Arora S.*, *Barak B.* Computational complexity: a Modern Approach. Cambridge; New York: Cambridge University Press, 2009. 579 p.

ОГЛАВЛЕНИЕ

Учебное издание

Ключарев Петр Георгиевич
Жуков Дмитрий Александрович

ВВЕДЕНИЕ В ТЕОРИЮ АЛГОРИТМОВ

Учебное пособие

Редактор *О.М. Королева*
Корректор *Е.В. Авалова*
Компьютерная верстка *О.В. Беляевой*

Подписано в печать 04.07.2012. Формат 60×84/16. Усл. печ. 2,33.
Изд. № 48. Тираж 100 экз. Заказ

Издательство МГТУ им. Н.Э. Баумана.
Типография МГТУ им. Н.Э. Баумана.
105005, Москва, 2-я Бауманская ул., 5.