Г.П. Агибалов

ТЕОРИЯ ВЫЧИСЛИТЕЛЬНОЙ СЛОЖНОСТИ

Учебное пособие

Министерство науки и высшего образования
Российской Федерации
Томский государственный университет

Г. П. Агибалов

ТЕОРИЯ ВЫЧИСЛИТЕЛЬНОЙ СЛОЖНОСТИ

Учебное пособие

Томск
2018

УДК 519.7
ББК 22.13я7

Агибалов Г. П. Теория вычислительной сложности: учеб. пособие. — Томск: Издательский Дом Томского государственного университета, 2018. — 42 с.

ISBN 978-5-94621-768-2

Пособие представляет собой курс лекций с тем же названием, прочитанный автором в 2016/17 и 2017/18 учебных годах студентам кафедры защиты информации и криптографии по специальности «Компьютерная безопасность». Знакомство с курсом предполагает знание студентами основ дискретной математики и начальных понятий алгебры, теории алгоритмов и математической логики.

УДК 519.7
ББК 22.13я7

ISBN 978-5-94621-768-2

Предисловие

Теория вычислительной сложности (ТВС) — это наука о сложности алгоритмов. Есть много книг о ней, в том числе исчерпывающего содержания. Наше учебное пособие этим свойством не обладает и не претендует на полное изложение ТВС в её современном состоянии. Оно не рассчитано и на профессиональных специалистов по теории вычислительной сложности. Даже название пособия не отражает его содержания. Оно просто повторяет название 36-часового курса лекций из образовательной программы Томского государственного университета по специальности Компьютерная безопасность. Его цель — дать студентам этой специальности начальные понятия из теории временно́й сложности алгоритмов и важнейшие результаты из теории разрешимости алгоритмических задач в худшем и типичном случаях, включая теоремы о NP-полноте задачи выполнимости, о генерической разрешимости задачи останова машины Тьюринга и о генерической сложности задачи дискретного логарифмирования.

С благодарностью

Валентине Владимировне Быковой — за материал по асимптотическим оценкам сложности и основным сложностным классам алгоритмов;

Александру Николаевичу Рыбалову — за материал по генерической сложности алгоритмов;

Ирине Анатольевне Панкратовой — за редактирование и подготовку рукописи лекций к публикации в данном формате.

Геннадий Петрович Агибалов
01.09.2018

1. Сложность алгоритмов

Сложность алгоритма — одна из мер его качества, а именно, мера эффективности алгоритма. К примеру, другой мерой качества алгоритма служит его точность. По любой мере качества можно сравнивать различные алгоритмы, решающие одну и ту же задачу. Можно говорить, например, что один алгоритм эффективнее другого, который, в свою очередь, точнее первого, т. е. первый алгоритм имеет меньшую сложность, но второй более точный — доставляемое им решение задачи в каком-либо смысле (по значению, по расстоянию) ближе к истинному решению и т. п. Здесь и далее под задачей понимается то, что в английском языке называется problem, а в русском иногда — массовой проблемой, или множеством экземпляров проблемы, решаемых одним и тем же алгоритмом.

Понятие сложности алгоритма предполагает наличие у решаемой им задачи числовой характеристики, называемой *размером* задачи и являющейся в действительности размером входных данных (или, как говорят, *входа*) алгоритма, выражающим фактически меру их количества. Например, размером задачи сравнения слов служит наибольшая длина сравниваемых слов. Размером задачи умножения чисел в какой-либо системе счисления является наибольшее количество знаков в числе-сомножителе, представленном в этой системе. Аналогично, размером задачи умножения матриц может быть наибольший размер матрицы, участвующей в умножении. Количество рёбер в графе или слагаемых в АНФ булевой функции может служить размером задачи о графах или булевых функциях соответственно.

Время, рассматриваемое как некоторая функция $t(n)$ от размера n задачи и затрачиваемое некоторым алгоритмом A на решение этой задачи, называется *временнóй сложностью* алгоритма A. По определению, $t(n)$ — это число единиц времени, требуемое алгоритму A для обработки входа размера n. Поскольку это число может зависеть не только от размера входных данных, но и от их значения, допускают некоторые разновидности временной сложности алгоритмов, в том числе сложность в среднем, сложность в худшем случае, сложность в типичном случае и др. Временная сложность алгоритма в среднем — это $t(n) = \dfrac{1}{|I_n|} \sum\limits_{i \in I_n} t_i$ —

результат усреднения времени t_i, требуемого алгоритму на обработку входа i, на множестве I_n всех входных данных размера n. Временная сложность алгоритма в худшем случае — это $t(n) = \max\limits_{i \in I_n} t_i$. Временная сложность алгоритмов в типичном случае, или на большинстве входов, называемая также генерической сложностью, определяется и рассматривается отдельно в последнем разделе. До того всюду под временной сложностью и под $t(n)$ подразумевается временная сложность алгоритма в худшем случае. Предел этой функции $\lim\limits_{n \to \infty} t(n)$, если он существует, называется *асимптотической временной сложностью* алгоритма A.

Аналогично определяются *ёмкостная сложность* алгоритма и её разновидности. При этом под ёмкостью подразумевается размер памяти, затрачиваемой алгоритмом для представления (записи, хранения) входных, выходных и промежуточных данных в процессе решения задачи.

Можно говорить также и о так называемой *параметрической* временной (или ёмкостной) сложности алгоритма и о её разновидностях в разных случаях, понимая под нею временную (или соответственно ёмкостную) его сложность, зависящую от размеров входов двух разных типов — информационного (n) и параметрического (k), и оцениваемую в каждом рассматриваемом случае в зависимости только от первого при фиксированном значении второго. Например, в алгоритмах на графах в роли первого может выступать число вершин или рёбер в графе, а в роли второго — степень вершины, диаметр графа и т. п.; в алгоритмах на булевых функциях n — это число переменных, а k — это степень нелинейности, мощность линеаризационного множества, количество мономов в АНФ; в криптосистемах шифрования n — это размер шифруемого текста, k — размер ключа, на котором производится зашифрование, и др. Но об этом — в другой раз и в другом месте.

В случае, если алгоритм представлен на некотором алгоритмическом языке, в качестве синонима временной сложности алгоритма употребляют его *вычислительную сложность*, понимаемую как количество операций этого языка, выраженное некоторой функцией от размера задачи, которые алгоритм выполняет, решая задачу (каждая операция считается столько раз, сколько раз она выполняется). Говорят также и об асимптоти-

ческой вычислительной сложности. Ёмкостная сложность алгоритма в этом случае определяется как сумма размеров памяти, требуемой алгоритмом для выполнения всех операций языка в нём.

В частности, если алгоритм описывается как машина Тьюринга (далее МТ), то говорят, что МТ M имеет временную сложность $t(n)$, если на любом её входе длины n она делает не более $t(n)$ переходов и останавливается.

Приведём пример из [1, с. 12–13], демонстрирущий тот факт, что для увеличения размера решаемой задачи важнее не увеличение скорости работы компьютера, а уменьшение временной сложности алгоритма решения этой задачи. Предположим, что единицей измерения времени является одна миллисекунда. Тогда максимальный размер задачи, решаемый за 1 с алгоритмами с временной сложностью n^3 и n, равен соответственно 10 и $1000 = 10^3$, т. е. замена первого алгоритма (с большей временной сложностью) более быстрым вторым алгоритмом увеличивает размер решаемой задачи в 100 раз, в то время как повышение скорости компьютера в 10 раз увеличивает максимальный размер задачи, решаемой первым алгоритмом за то же время, только в 2,15 раза. Различие становится более значительным, если сравнение делается на большем промежутке рабочего времени.

2. Асимптотические оценки сложности алгоритмов

Под асимптотической оценкой сложности $t(n)$ понимается некоторая неотрицательная вещественная функция $s(n)$, имеющая несложное (арифметическое) выражение через n, поведение (порядок роста) которой с ростом n (при $n \to \infty$) совпадает, возможно, с какой-то точностью с ростом функции $t(n)$, за исключением значений коэффициентов и аддитивных членов неглавных порядков в ней. В этом разделе мы рассмотрим возможные разновидности таких оценок — сверху, снизу, точные. Для упрощения записи обозначим $d(n) = \dfrac{t(n)}{s(n)}$ и $lim = \lim\limits_{n\to\infty} d(n)$, если этот предел существует.

Будем говорить, что $t(n)$ *меньше функции* $s(n)$ *по порядку роста*, и писать $t(n) \prec s(n)$, или $t(n) = \mathrm{o}(s(n))$, если $lim = 0$. В этом случае $s(n)$ называется о-*оценкой* функции $t(n)$. В частности, за-

пись $t(n) = \mathrm{o}(1)$ означает, что $t(n)$ стремится к 0, становясь сколь угодно малой, при $n \to \infty$.

В случае $lim = c > 0$ говорят, что функции $t(n)$ и $s(n)$ *асимптотически пропорциональны*, или имеют *один и тот же порядок роста*, и пишут $t(n) = \mathrm{O}^*(s(n))$, или $t(n) \sim cs(n)$. В этом случае $s(n)$ называется O^*-*оценкой* функции $t(n)$. В частности, если $t(n) = \mathrm{O}^*(1)$, то $t(n)$ называют *асимптотической константой*. В случае $c = 1$, т. е. когда $t(n) \sim s(n)$, говорят, что $t(n)$ *эквивалентна, или асимптотически равна, функции* $s(n)$.

Говорят, что функция $t(n)$ *не больше функции* $s(n)$ *по порядку роста*, и пишут $t(n) = \mathrm{O}(s(n))$, или $t(n) \preccurlyeq s(n)$, если их отношение $d(n)$ *ограничено сверху*, т. е. существуют такие натуральное число n_0 и константа $c_1 > 0$, что для всех $n \geqslant n_0$ имеет место неравенство $d(n) \leqslant c_1$. В этом случае $s(n)$ называют *асимптотической верхней оценкой*, или O-*оценкой* для $t(n)$.

Двойственным образом определяется *асмптотическая нижняя оценка* для $t(n)$, а именно: если отношение $d(n)$ *ограничено снизу*, т. е. существуют такие натуральное число n_0 и константа $c_2 > 0$, что для всех $n \geqslant n_0$ имеет место неравенство $d(n) \geqslant c_2$, то говорят, что $t(n)$ *не меньше функции* $s(n)$ *по порядку роста*, и пишут $t(n) = \Omega(s(n))$, или $s(n) \preccurlyeq t(n)$. В этом случае $s(n)$ называется Ω-*оценкой* функции $t(n)$.

Наконец, если отношение $d(n)$ функций $t(n)$ и $s(n)$ ограничено одновременно и сверху и снизу, т. е. $c_2 \leqslant d(n) \leqslant c_1$ для некоторых $c_1 > 0$ и $c_2 > 0$, то пишут $t(n) = \Theta(s(n))$, или $t(n) \asymp s(n)$, говорят, что $t(n)$ *равна функции* $s(n)$ *по порядку роста*, и называют $s(n)$ *асимптотически точной оценкой*, или Θ-*оценкой* функции $t(n)$. Следует заметить, что если предел $\lim\limits_{n\to\infty} d(n) = c > 0$ существует и, следовательно, существует O^*-оценка функции $t(n)$, то эта оценка совпадает с Θ-оценкой для $t(n)$ и также является асимптотически точной оценкой для $t(n)$, т. е. $t(n) \asymp s(n)$. Что касается Θ-оценки, то она может существовать и без существования O^*-оценки.

Получение именно асимптотически точных оценок желательно для функций сложности алгоритмов. Когда этого не удаётся сделать, строят асимптотические оценки других видов.

Остановимся на некоторых свойствах оценок $\mathrm{o}, \mathrm{O}^*, \mathrm{O}, \Omega, \Theta$, вытекающих непосредственно из их определений и являющихся

фактически свойствами соответствующих бинарных отношений $\prec, \sim, \preceq, \succeq, \asymp$. Пусть далее $t(n)$, $s(n)$, $s_1(n)$, $s_2(n)$ и $u(n)$ — неотрицательные вещественные функции, не обращающиеся в 0, по крайней мере, для $n \geqslant n_0$.

1. Свойства эквивалентности:

1) рефлексивность — $s(n) = \varphi(s(n))$ для $\varphi \in \{\mathrm{O}^*, \mathrm{O}, \Omega, \Theta\}$;
2) симметричность — $t(n) = \varphi(s(n)) \Rightarrow s(n) = \varphi(t(n))$ для $\varphi \in \{\mathrm{O}^*, \Theta\}$;
3) транзитивность — $\big(t(n) = \varphi(s(n))\ \&\ s(n) = \varphi(u(n))\big) \Rightarrow$ $\Rightarrow \big(t(n) = \varphi(u(n))\big)$ для всех $\varphi \in \{\mathrm{o}, \mathrm{O}^*, \mathrm{O}, \Omega, \Theta\}$.

2. Свойства включения:

1) $t(n) = \mathrm{o}(s(n)) \Rightarrow t(n) = \mathrm{O}(s(n))$, обратное неверно;
2) $t(n) = \Theta(s(n)) \Leftrightarrow \big(t(n) = \mathrm{O}(s(n))\ \&\ t(n) = \Omega(s(n))\big)$;
3) $t(n) = \mathrm{O}(s(n)) \Rightarrow s(n) = \Omega(t(n))$;
4) $t(n) = \Omega(s(n)) \Rightarrow s(n) = \mathrm{O}(t(n))$;
5) $t(n) = \mathrm{O}^*(s(n)) \Rightarrow t(n) = \Theta(s(n))$.

3. Правила арифметических операций для $\varphi \in \{\mathrm{O}, \Theta\}$:

1) правило суммы — $\big(t(n) = \varphi(s_1(n))\ \&\ u(n) = \varphi(s_2(n))\ \&$ $\&\ s_2(n) \prec s_1(n)\big) \Rightarrow \big(t(n) + u(n) = \varphi(s_1(n))\big)$; в частности, $\varphi(t(n) + c) = \varphi(t(n))$ для любой константы $c > 0$;
2) правило произведения — $\big(t(n) = \varphi(s_1(n))\ \&\ u(n) = \varphi(s_2(n))\big) \Rightarrow$ $\Rightarrow \big(t(n) \cdot u(n) = \varphi(s_1(n) \cdot s_2(n))\big)$; в частности, $t(n) \cdot c =$ $= \varphi(t(n))$ для любой константы $c > 0$.

3. Основные сложностные классы алгоритмов

Пусть $t_1(n)$ и $t_2(n)$ суть функции сложности алгоритмов A_1 и A_2 соответственно. Считают, что алгоритм A_1 *асимптотически быстрее* алгоритма A_2, если $t_1(n) \prec t_2(n)$, т. е. $t_1(n) =$ $= \mathrm{o}(t_2(n))$; алгоритм A_1 *асимптотически не медленнее* алгоритма A_2, если $t_1(n) \preccurlyeq t_2(n)$, т. е. $t_1(n) = \mathrm{O}(t_2(n))$; алгоритм A_1 *асимптотически не быстрее* алгоритма A_2, если $t_2(n) \preccurlyeq t_1(n)$, т. е. $t_1(n) = \Omega(t_2(n))$; наконец, алгоритмы A_1 и A_2 *одной асимптотической сложности*, или *асимптотически равнозначны по сложности*, если $t_1(n) \asymp t_2(n)$, т. е. $t_1(n) = \Theta(t_2(n))$.

В анализе конкретного алгоритма всегда стремятся получить асимптотически точную оценку его сложности — Θ-оценку. В случае неудачи ограничиваются обычно O-оценкой. В классификации алгоритмов по вычислительной сложности важную

роль играют и другие оценки, позволяющие выстроить достаточно полную иерархию сложностных классов алгоритмов с наполненным их содержанием.

Представляется естественным все алгоритмы разделить сначала на два класса — *эффективные*, т. е. те, что на реальных входных данных (реального размера n) могут быть выполнены за реальное время $t(n)$, и *неэффективные* — все остальные, те, что на реальных входах требуют для своего выполнения нереального времени — того, которого у вычислителя в действительности нет. В науке об алгоритмах принято к первым относить алгоритмы с вычислительной сложностью $t(n) = \mathrm{O}(n^c)$, или $t(n) \preceq n^c$, где c — неотрицательное действительное число, и называть их алгоритмами *полиномиальной сложности*, или *полиномиальными алгоритмами*, а ко вторым — все алгоритмы, функции сложности которых растут быстрее любого полинома от n, т. е. алгоритмы с вычислительной сложностью $t(n) \succ n^c$ для всех $c > 0$, и называть их алгоритмами *экспоненциальной сложности*, или *экспоненциальными алгоритмами*.

Класс полиномиальных алгоритмов замкнут относительно операции композиции и полиномиального преобразования их входов. Точно так же, композиция алгоритмов, среди которых есть экспоненциальный алгоритм, является алгоритмом экспоненциальным, а любое преобразование входа экспоненциального алгоритма оставляет алгоритм экспоненциальным.

Строго говоря, функция сложности полиномиального алгоритма может отличаться от полинома, а функция сложности экспоненциального алгоритма — от экспоненты. Так, в класс полиномиальных алгоритмов входят алгоритмы с функцией сложности $t(n) = \Theta(\log_2 n)$, а в класс экспоненциальных алгоритмов — алгоритм с функцией сложности $t(n) = \Theta(n^2 \cdot n!)$. Последнее — ввиду того, что $n^c \prec \mathrm{e}^{\lambda n} \prec n!$ для $\lambda > 0$, $c > 0$. В этой связи в классе полиномиальных алгоритмов выделяют субполиномиальные, а в классе экспоненциальных — субэкспоненциальные и гиперэкспоненциальные алгоритмы. В переводе с латинского префикс *суб* означает *под* или *около*, а префикс *гипер* — *над* или *сверх*.

Время работы *субполиномиального* алгоритма растёт медленнее любого полинома n^c, $c > 0$, а время работы *субэкспоненциального* алгоритма растёт медленне любой экспоненты $\mathrm{e}^{\lambda n}$, $\lambda > 0$,

но, разумеется, быстрее любого полинома. Функция сложности собственно полиномиального алгоритма, не являющегося субполиномиальным, имеет порядок роста $\Theta(n^c)$.

Приведём примеры алгоритмов данных классов вычислительной сложности. В них n — количество разрядов в двоичном представлении целого числа.

Субполиномиальные алгоритмы: бинарный поиск заданного элемента в отсортированном массиве из m чисел, имеет сложность $t(m) = \Theta(\log_2 m)$; алгоритм определения числа точек эллиптической кривой над полем $\mathrm{GF}(p)$, $t(p) = \mathrm{O}((\log_2 p)^6)$.

Полиномиальные алгоритмы: алгоритм Евклида для вычисления наибольшего общего делителя целых чисел, $t(n) = \mathrm{O}(n^3)$; алгоритм Штрассена исключения переменной в системе из m линейных уравнений над конечным полем, $t(m) = \mathrm{O}(m^{\log_2 7})$.

Субэкспоненциальный алгоритм: алгоритм решета числового поля для факторизации n-разрядного целого числа, $t(n) = \mathrm{e}^{n^{1/3}(\ln n)^{2/3}} = \mathrm{o}(\mathrm{e}^n)$.

Экспоненциальные алгоритмы: алгоритмы факторизации Полларда — Штрассена и Ленстры, $t(n) = \Theta(\mathrm{e}^{(n/4)n^3})$, $t(n) = \Theta(\mathrm{e}^{(n/3)n^2})$.

Гиперэкспоненциальный алгоритм: распознавание гамильтоновости графа с m вершинами, $t(m) = \Theta(m^2 \cdot m!)$.

4. Алгоритмические задачи

Так мы называем задачи, решаемые алгоритмами. Они бывают *функциональными*, или *function problems*, и *логическими*, или *decision problems*. Формулировки первых начинаются со слов «найти», «получить», «определить», «вычислить» и т. п., а их ответами служат значения некоторых функций, принадлежащие некоторым множествам. Формулировки вторых начинаются со слов «существует ли», «верно ли, что», «узнать, что», «является ли» и т. п., а их ответами служат истинностные значения некоторых высказываний — «да» или «нет», «true» или «false».

Все логические задачи, требующие ответа «да» или «нет», объединяет одно свойство — они являются вопросом разрешения: истинно нечто или нет? Обычно их так и называют — задачами резрешения, а алгоритмы их решения — разрешающими алгоритмами. В функциональных же задачах требуется «найти решение». Они являются вопросом вычисления: чему равно

нечто? Обычно их так и называют — задачами вычисления, или *computation problems*, а алгоритмы их решения — вычислительными алгоритмами.

Разумеется, такое разделение алгоритмических задач на функциональные и логические не является чистым, ибо, например, задачи математической логики, заключающиеся в нахождении истинностного значения булева выражения для заданных значений переменных в нём являются одновременно и вычислительными, так как ответ в них равен значению «true» или «false» некоторой булевой функции, и задачами разрешения, так как этот ответ является одновременно и ответом на вопрос, истинно ли данное выражение — «да» или «нет»?

Примеры функциональных задач: решение системы уравнений, вычисление логарифма в группе, факторизация целых чисел, нахождение в графе клики максимального размера, шифрование, цифровое подписание и многие другие. Здесь же и многочисленные задачи подсчёта, перечисления, разбиения, поиска, оптимизации.

Примеры лгических задач: проблема останова машины Тьюринга (алгоритма, программы), проверка целого числа на простоту (системы уравнений на совместность, булевой формулы на выполнимость), аутентификация сообщения, идентификация личности, контроль целостности сообщения, проверка наличия в графе клики заданного размера, проблема принадлежности подмножеству некоторого множества и многие другие. Здесь же и знаменитая проблема соответствия Поста, состоящая в том, что требуется узнать, есть ли в произвольной конечной последовательности пар слов $(x_1, y_1)(x_2, y_2)\dots(x_m, y_m)$ в некотором алфавите подпоследовательность $(x_{i_1}, y_{i_1})(x_{i_2}, y_{i_2})\dots(x_{i_k}, y_{i_k})$, где $1 = i_1 < i_2 < \ldots < i_k$, $k \leqslant m$, такая, что имеет место равенство конкатенаций $x_{i_1}x_{i_2}\dots x_{i_k} = y_{i_1}y_{i_2}\dots y_{i_k}$.

Некоторые из функциональных задач несложно преобразуются в задачи логические. Так, задача нахождения наибольшей клики в графе G, относящаяся к классу функциональных оптимизационных, допускает преобразование за полиномиальное время от длины n кодового представления графа G к логической задаче о наличии в G клики заданного размера. В самом деле, имея алгоритм решения последней, выясним последовательно, есть ли в G клики мощности 1, 2, …, и таким образом найдём размер m

наибольшей клики в G. Удаляя из G по одной вершине (вместе с инцидентными ей рёбрами), найдём первую такую вершину v, после удаления которой на месте G остаётся граф без клик мощности m. Рассмотрим в G подграф G', порождённый вершинами, смежными с вершиной v. Применяя рекурсивно данную процедуру, найдём в G' клику C мощности $m-1$. После этого наибольшая клика в G есть подграф, порождённый вершинами в C и v.

В теории сложности алгоритмов фундаментальное значение имеет логическая задача принадлежности подмножеству, в которой для произвольного элемента некоторого множества требуется узнать, принадлежит ли он заданному подмножеству этого множества. Часто последним служит множество всех слов в некотором алфавите, а подмножеством — язык в нём, и тогда говорят о задаче принадлежности к языку. Для её точной постановки напомним необходимые определения. Любое конечное линейно упорядоченное множество Σ называется *алфавитом*, его элементы — *буквами*, *знаками* или *символами*. Последовательности конечной длины, составленные из букв алфавита, называются *словами* в этом алфавите. Любое множество L слов в алфавите Σ называется *языком* в данном алфавите. Множество всех слов в алфавите Σ обозначается Σ^*. Это есть универсальный язык в алфавите Σ, содержащий в себе все языки в этом алфавите.

Конечный язык может быть задан перечислением своих слов; другие языки могут быть заданы при помощи индуктивных определений, где база индукции фиксирует некоторые первичные слова в языке, а шаг индукции содержит правила образования других слов языка из уже имеющихся в нём слов; или, скажем, посредством предиката P с предметной областью Σ^* по правилу $L = \{w : P(w) = \text{true}\}$.

Задача принадлежности языку формулируется так: даны слово w из Σ^*, называемое *экземпляром* задачи, и язык $L \subset \Sigma^*$, называемый её языком; требуется выяснить, принадлежит слово w языку L или нет.

Причина интереса теоретиков к данной задаче двоякая — привлекающая простота её формулировки и её универсальность, а именно: всё, что мы называем здесь задачей как массовой проблемой, может быть выражено в виде задачи о принадлежности слова языку. Разумеется, эту её универсальность в отсутствие

формального определения понятия задачи невозможно доказать, но за неё — весь наш исследовательский опыт: многие алгоритмические задачи из разных областей дискретной математики (о булевых функциях, графах, целых числах, автоматах, кодах, алгебраических и криптосистемах) допускают формулировки как задачи принадлежности языку.

Например, если задача P логическая и, следовательно, её ответом при каждом значении входных данных может быть только «да» или «нет», то, закодировав все возможные значения её входов словами в некотором алфавите Σ и отнеся к языку L коды тех входных данных, на которых ответ задачи P есть «да», получим формулировку задачи о принадлежности слова из Σ^* языку L. Так можно задачу проверки целого числа на простоту выразить в терминах принадлежности двоичного кода целого числа к языку из двоичных кодов простых чисел, а задачу выяснения наличия в графе клики заданной мощности сформулировать как задачу принадлежности последовательности бит в матрице смежности графа к языку последовательностей бит в матрицах смежности графов, содержащих такие клики. Наконец, задачу проверки выполнимости конъюнктивной нормальной формы (КНФ) булевой функции можно привести к задаче принадлежности к языку, если любую КНФ закодировать последовательностью троичных векторов, представляющих стандартным образом элементарные дизъюнкции в ней, и в качестве языка взять множество кодов всех выполнимых КНФ.

Если же задача P функциональная, её предварительно преобразуют к логической задаче, как это было продемонстрировано на задаче о максимальной клике графа, а затем — к задаче принадлежности к языку, как описано только что.

Задание языка L однозначно определяет задачу принадлежности к нему и наоборот. В этой связи задачи принадлежности к языкам принято отождествлять с их языками и там, где это не вызывает недоразумений, называть язык L задачей L, или L-задачей, понимая под ней именно задачу принадлежности к языку L.

Условимся впредь для задач P_1 и P_2 принадлежности к языкам $L_1 \subset \Sigma_1^*$ и $L_2 \subset \Sigma_2^*$ соответственно наряду с $w_1 \in L_1 \Leftrightarrow$ $\Leftrightarrow w_2 \in L_2$ писать $L_1(w_1) = L_2(w_2)$, или $P_1(w_1) = P_2(w_2)$. Будем говорить также, что задача P_1 *сводима* к задаче P_2, если суще-

ствует алгоритм $A : \Sigma_1^* \to \Sigma_2^*$, преобразующий слова $w_1 \in \Sigma_1^*$, являющиеся экземплярами P_1, в слова $w \in \Sigma_2^*$, являющиеся экземплярами P_2, так, что если $A(w_1) = w$, то $w_1 \in L_1 \Leftrightarrow w \in L_2$, т. е. $P_1(w_1) = P_2(A(w_1))$ для любого $w_1 \in \Sigma_1^*$.

В дальнейшем, в отсутствие дополнительных оговорок, под задачей всякий раз подразумевается задача принадлежности некоторому языку.

5. Алгоритмы разрешения

Пусть I есть множество и L — его подмножество. Обозначим $D = (L, I)$ задачу разрешения, в которой требуется для каждого входа $x \in I$ решить, принадлежит x подмножеству L или нет. В частности, если I есть множество слов в некотором алфавите и L — язык в нём, то D является задачей принадлежности этому языку. Алгоритм $\mathcal{A}$, который для каждого своего входа $x \in I$ выдаёт ответ 1 («да») ($\mathcal{A}(x) = 1$), если $x \in L$, и ответ 0 («нет») ($\mathcal{A}(x) = 0$), если $x \notin L$, называется *разрешающим*, или *алгоритмом разрешения*, для D. В случае его существования подмножество L называется *разрешимым*. Множество всех разрешимых подмножеств во всяком I замкнуто относительно операций объединения, пересечения и дополнения.

Для любого подмножества $S \subseteq I$ задача разрешения $D_S = (L \cap S, S)$ называется *частью*, или *ограничением*, задачи D (на подмножестве S). Разрешающий алгоритм для D_S называется *частичным разрешающим* алгоритмом для D (на S).

Теорема 5.1. Если для D существует разрешающий алгоритм $\mathcal{A}$, то для любого $S \subseteq I$ существует разрешающий алгоритм $\mathcal{B}$ для D_S.

Доказательство. Для каждого $x \in S$ положим $\mathcal{B}(x) = \mathcal{A}(x)$. Пусть $x \in S$. Тогда $x \in I$ и $x \in L \cap S \Leftrightarrow x \in L \Leftrightarrow \mathcal{A}(x) = 1 \Leftrightarrow \Leftrightarrow \mathcal{B}(x) = 1$. ■

Следствие 1. Если для некоторого $S \subseteq I$ не существует разрешающего алгоритма $\mathcal{B}$ для D_S, то не существует разрешающего алгоритма $\mathcal{A}$ для D.

Доказательство. От противного. ■

Говорят, что алгоритм $\mathcal{A}$ *допускает* вход $x \in I$, если через конечное число шагов после начала работы над x он останав-

ливается. Множество всех входов, допускаемых алгоритмом $\mathcal{A}$, называется его множеством *останова* и обычно обозначается $H_{\mathcal{A}}$ (по первой букве в слове halting — останов).

Всякое разрешимое подмножество является множеством останова некоторого алгоритма, и существуют неразрешимые множества останова алгоритмов.

6. Неразрешимые задачи

Задача называется *неразрешимой*, или *нерешаемой*, если не существует алгоритма её разрешения; в противном случае задача *разрешима* (*решаемая*). Легко убедиться, что почти все задачи являются неразрешимыми в любой формальной системе представления алгоритмов, включая машину Тьюринга и программирование. Это следует уже из того, что множество всех алгоритмов в системе как слов в некотором алфавите счётное (его элементы можно перенумеровать натуральными числами, например, в лексикографическом порядке), а множество всех задач принадлежности слов к различным языкам в этом алфавите несчётное (ввиду несчётности множества самих языков). Таким образом, задач существует «бесконечно больше», чем алгоритмов. Если случайно выбрать задачу, то наверняка она окажется неразрешимой, а то, что большинство встречающихся задач разрешимые, объясняется нашим редким обращением к случайным задачам: нам свойственно изучать простые и хорошо структурированные задачи, а они действительно часто разрешимы.

Бывает нелегко доказать, что какая-то конкретная задача P неразрешимая. Иногда это удаётся сделать методом от противного: допустить существование решающего алгоритма и путём логических рассуждений придти к некоторому противоречию.

Другой метод доказать, что задача P неразрешима, состоит в том, чтобы свести к ней известную неразрешимую задачу — P_1. В самом деле, если неразрешимая задача P_1 сводится к задаче P, то для любого экземпляра w_1 первой верно $P_1(w_1) = P(A_1(w_1))$ для некоторого алгоритма A_1, из чего следует, что если задача P разрешима и, значит, решается некоторым алгоримом A, то задача P_1 решается композицией алгоритмов A_1 и A и, значит, тоже разрешима, что не так; следовательно, P неразрешима. Таким образом, для доказательства неразрешимости какой-то задачи P достаточно свести к ней какую-либо неразрешимую задачу

P_1. Метод так и называется — *сведение* одной задачи (P_1) к другой (P).

Примеры неразрешимых задач: проблема останова машины Тьюринга (алгоритма, программы), проблема соответствия Поста.

7. Трудноразрешимые задачи

Все разрешимые задачи можно разделить на разрешимые за полиномиальное время, т. е. решаемые алгоритмами с полиномиальной или меньшей временной сложностью (за полиномиальное или меньшее время), и на разрешимые за экспоненциальное или большее время, т. е. требующие для своего решения алгоритмов с экспоненциальной и большей временной сложностью. Практика показывает, что это разделение алгоритмических задач фундаментальное: полиномиальное время решения задач является, как правило, практически приемлемым, в то время как экспоненциальное время таковым не является — задачи, требующие такого времени, практически (за приемлемое время) неразрешимы. Эти, последние, задачи и называются *трудноразрешимыми*, или просто трудными.

Теория трудноразрешимости занимается изучением методов доказательства неразрешимости задач за полиномиальное время, называемой также *полиномиальной неразрешимостью*. Как и в теории неразрешимости, основным методом здесь является сведение одной задачи (в данном случае — полиномиально неразрешимой) к другой, полиномиальную неразрешимость которой требуется доказать. Но теперь уже недостаточно просто наличия алгоритма, переводящего экземпляры одной задачи в экземпляры другой. Этот алгоритм обязательно должен иметь полиномиальную временную сложность, так как время, затрачиваемое на такой перевод, практически затрачивается на получение решения второй задачи, и, будь оно экспоненциальным, оно не даст нам права считать эту задачу трудноразрешимой, поскольку такое сведение к ней полиномиально неразрешимой задачи не исключает её собственной полиномиальной разрешимости.

Есть ещё одно принципиальное различие между доказательствами в современных теориях неразрешимости и трудноразрешимости. Доказательства (неразрешимости) в первой основаны на точном определении алгоритма — как машина Тьюринга, на-

пример, и на уже доказанных предпосылках и потому неопровержимы. Доказательства же (полиномиальной неразрешимости) во второй теории основаны на недоказанном предположении $\mathcal{P} \neq \mathcal{NP}$, где $\mathcal{P}$ и $\mathcal{NP}$ обозначают классы задач, разрешимых за полиномиальное время соответственно детерминированными и недетерминированными машинами Тьюринга. Поскольку первые являются частными случаями вторых, имеет место включение $\mathcal{P} \subseteq \mathcal{NP}$, но обратное включение пока и не доказано, и не опровергнуто.

8. Машины Тьюринга

О классах $\mathcal{P}$ и $\mathcal{NP}$ мы ещё поговорим, а сейчас, в порядке напоминания, приведём некоторые нужные нам сведения о машинах Тьюринга. Каждая МТ содержит в себе несколько (одну или более) лент, бесконечных в обе стороны или только вправо, разбитых на клетки (ячейки) для записи на них слов в некотором алфавите — по одному символу в клетке. Предполагается, что запись на пустую ленту всегда начинается с клетки, принимаемой за первую. В односторонней ленте это самая левая клетка. Каждая лента снабжена подвижной головкой для обозрения на ленте любой клетки с целью записи в неё или считывания из неё символа. В каждый момент дискретного времени положение головки на ленте и выполняемые ею операции (запись и считывание) определяются управляющим устройством, в роли которого выступает конечный автомат.

МТ бывают детерминированные (ДМТ) и недетерминированные (НМТ). Формально *k-ленточная НМТ* M есть $M = (A, X, Q, q_0, q_f, \delta)$, где A, X и Q — конечные множества, называемые соответственно *входным алфавитом* (с входными символами), *ленточным алфавитом* (с ленточными символами) и *множеством состояний* (управляющего автомата), $A \subseteq X$, $X \cap Q = \varnothing$; $q_0 \in Q$ — *начальное состояние*; $q_f \in Q$ — *заключительное состояние* и $\delta : Q \times X^k \to 2^{Q \times (X \times \{L,R,S\})^k}$ — *функция переходов* управляющего автомата, которая по его состоянию $q \in Q$ и набору $x = x_1 x_2 \ldots x_k$ символов x_j в клетках, обозреваемых головками на лентах с номерами $j = 1, 2, \ldots, k$ соответственно, вычисляет подмножество $\delta(q, x) = \{(s^{(i)}, (y_1^{(i)}, d_1^{(i)}), (y_2^{(i)}, d_2^{(i)}), \ldots, (y_k^{(i)}, d_k^{(i)})) : i = 1, 2, \ldots, m\}$, состоящее из некоторого числа $m \geqslant 0$ *вариантов перехода*, где i-й вариант

задаётся тройкой $(s^{(i)}, y^{(i)}, d^{(i)})$, содержащей новое состояние автомата $s^{(i)}$, набор $y^{(i)} = y_1^{(i)} y_2^{(i)} \dots y_k^{(i)}$ новых символов для записи в обозреваемые клетки на лентах и набор $d^{(i)} = d_1^{(i)} d_2^{(i)} \dots d_k^{(i)}$ направлений сдвига ленточных головок на одну клетку так, что головка j-й ленты сдвигается влево, если $d_j^{(i)} = L$, или вправо, если $d_j^{(i)} = R$, или остаётся на месте, если $d_j^{(i)} = S$, $j = 1, 2, \dots, k$. Количество m вариантов в каждом переходе (для каждой пары $(q, x) \in Q \times X^k$) своё, т. е. числа $|\delta(q, x)|$ и $|\delta(q', x')|$ не обязаны совпадать, если $(q, x) \neq (q', x')$. Ленточный алфавит X включает в себя, кроме входных символов, некоторые вспомогательные символы, среди которых Λ — *пустой символ*, или пробел. Впредь состояния управляющего автомата в МТ называются также состояниями этой МТ.

Предполагается, что все варианты перехода M, содержащиеся в значении $\delta(q, x)$, осуществляются параллельно, т. е. одновременно m копиями, или «клонами» $M^{(1)}, \dots, M^{(m)}$ машины M так, что для каждого $i = 1, \dots, m$ клон $M^{(i)}$ из состояния q переходит в состояние $s^{(i)}$, на его j-й ленте символ x_j в клетке, обозреваемой головкой ленты, заменяется символом $y_j^{(i)}$, и её головка сдвигается влево или вправо или не сдвигается в соответствии со значением $d_j^{(i)}$. Совокупность этих действий всех клонов естественно назвать одним шагом работы НМТ M.

Если говорить о функционировании НМТ M в дискретном времени $t = 1, 2, \dots$, то считается, что сначала M устанавливается в состояние q_0, на первую ленту, начиная с первой её клетки, записывается входное слово $w \in A^n$ некоторой длины $n \geqslant 1$, а на остальные ленты — пустое слово (из пустых символов), и головка каждой ленты устанавливается напротив её первой клетки. В произвольный момент времени $t \geqslant 1$ подразумевается, что имеется конечное множество клонов машины M: $M^{(i)}(t), i = 1, 2, \dots, N_t$, и каждый клон $M^{(i)}(t)$ находится в своём состоянии $s^{(i)}(t)$, имеет на своей j-й ленте некоторое слово $w_j^{(i)}(t)$ и головку, обозревающую некоторую клетку на ней с записанным в ней некоторым символом. В частности (при $t = 1$), имеем $N_1 = 1$, $M^{(1)}(1) = M$, $s^{(1)}(1) = q_0$, $w_1^{(1)}(1) = w$, $w_1^{(j)}(1)$ — пустое слово для $1 < j \leqslant k$, и головка каждой ленты обозревает первую клетку.

В момент времени t все клоны $M^{(i)}(t)$ выполняют свои переходы параллельно, как описано выше. Называя совокупность всех этих переходов t-м шагом работы МТ M над входным словом w, можно сказать, что $s^{(i)}(t)$ и $w_j^{(i)}(t)$ для $i = 1, 2, \ldots, N_t$ — это, соответственно, все те состояния, в которых МТ M может оказаться после $(t-1)$ шагов своей работы над w, и все те слова, которые могут оказаться на j-й ленте МТ M после этой её работы. Слово w называем *допускаемым* машиной M и говорим, что M *допускает* w, если $s^{(i)}(t) = q_f$ для некоторых $t \geqslant 1$ и $i \leqslant N_t$. Для наименьшего t, при котором это происходит для некоторого i, число $(t-1)$ обозначаем $t(w)$ и говорим, что M *допускает* w *со сложностью* $t(w)$. Множество всех слов в алфавите A, допускаемых M, обозначается $L(M)$ и называется *языком, допускаемым МТ* M. Функция $t_M(n) = \max\limits_{w \in L(M), |w|=n} t(w)$ называется *временной сложностью* машины M. Это есть точная верхняя граница для количества шагов, за которые данная НМТ M устанавливает факт допускаемости ею любого входного слова длины n, которое она действительно допускает. Другими словами, если некоторое слово w длины n принадлежит языку $L = L(M)$, то M может доказать эту принадлежность, затратив не более $t_M(n)$ шагов своей работы над этим словом, выполняя на каждом шаге проверку одновременно всех достижимых на этом шаге состояний на заключительность. Впредь, если из контекста понятно, о сложности какой МТ M идёт речь, вместо $t_M(n)$ пишем $t(n)$.

В случае $\delta(q, x) = \varnothing$, т. е. $m = 0$, говорим, что значение $\delta(q, x)$ *не определено*. Если $|\delta(q, x)| \leqslant 1$ для всех $(q, x) \in Q \times X^k$, то МТ M называется *детерминированной*, или *ДМТ*, а если $k = 1$, то 1-*ленточной* МТ. В последнем случае x есть одиночный ленточный символ. В 1-ленточной ДМТ значение функции переходов $\delta(q, x)$ из состоянии q под действием считываемого ленточного символа x, если оно определено, есть тройка (s, y, d), где s — новое состояние; y — новый символ, записываемый в обозреваемую клетку ленты; d — направление движения головки. В 1-ленточной НМТ это значение есть множество таких троек $\{(s_1, y_1, d_1), (s_2, y_2, d_2), \ldots, (s_m, y_m, d_m)\}$.

Если для некоторой МТ $x_1 x_2 \ldots x_l$ есть слово на некоторой её ленте с пустыми клетками всюду справа и слева от него, q —

состояние МТ и r — номер клетки на ленте, которую обозревает её головка, $1 \leqslant r \leqslant l$, то слово $x_1x_2\dots x_{r-1}qx_r\dots x_l$ называется *конфигурацией этой ленты*. Конфигурацию пустой ленты МТ в состоянии q с головкой, обозревающей 1-ю клетку, записываем как слово из одной буквы — q. Для k-ленточной НМТ M без символа «;» в X строка $(C_1;C_2;\dots;C_k)$ называется *конфигурацией в* M, если C_j для каждого $j=1,2,\dots,k$ есть конфигурация j-й ленты в M и во все эти конфигурации входит одно и то же состояние МТ M.

Для k-ленточной НМТ M и её входного слова $w \in A^*$ определяется *дерево* $T(M,w)$, в котором вершинами являются конфигурации в M, рёбрами — переходы в M, корнем (вершиной 1-го яруса) служит начальная конфигурация $\alpha_0 = q_0w;q_0;\dots;q_0$, составленная из начальных конфигураций k лент, и из произвольной вершины $V = (C_1;\dots;C_{j-1};x_1x_2\dots x_{r-2}zqxx_{r+1}\dots x_l;C_{j+1};\dots;C_k)$ произвольного яруса, в которой на j-й ленте $z = x_{r-1}$, $x = x_r$, $\delta(q,x) = \{(s_1,y_1,d_1),(s_2,y_2,d_2),\dots,(s_m,y_m,d_m)\}$, выходят m рёбер этого яруса с концами в вершинах следующего яруса $V_i = (C_1;\dots;C_{j-1};x_1x_2\dots x_{r-2}d_i(z,q,x)x_{r+1}\dots x_l;C_{j+1};\dots;C_k)$, где $d_i(z,q,x) = s_izy_i$, если $d_i = L$; $d_i(z,q,x) = zy_is_i$, если $d_i = R$; и $d_i(z,q,x) = zs_iy_i$, если $d_i = S$, $i = 1,2,\dots,m$. Вершина V объявляется *концевой* (без исходящих из неё рёбер), или *листом*, если в ней состояние q заключительное или значение $\delta(q,x)$ не определено, или, для односторонней ленты, $r = 1$ и $d_i = L$. В первом случае вершину V называют также *заключительной*, или *допускающей*.

Ясно, что M допускает входное слово w, если в $T(M,w)$ есть путь от корня к концевой вершине с заключительным состоянием, и не допускает, если дерево $T(M,w)$ конечное (число ярусов в нём ограничено) и ни один из его листьев не является допускающей вершиной. В случае допускаемого слова w сложность его допускания $t(w)$ есть длина кратчайшего пути в $T(M,w)$ от корня к листу с заключительным состоянием. По построению длина ленточного слова в конфигурации любой ленты в какой-либо вершине $T(M,w)$ не превосходит величины $n+t$, где $n = |w|$ и t — длина пути к этой вершине от корня, а головка отстоит от первой клетки не далее, чем на t клеток. Ясно также, что $T(M,w)$ для ДМТ M не имеет ветвлений, в нём есть только один путь из корня дерева.

Теорема 8.1. Для любой НМТ M_N с временной сложностью $t(n)$ существует ДМТ M_D с временной сложностью $\mathrm{O}(c^{t(n)})$, такая, что $L(M_D) = L(M_N)$.

Доказательство. Требуемая M_D моделирует обход в глубину максимального поддерева T' с высотой $t(n)$ в дереве конфигураций $T(M_N, w)$ и с его корнем и по достижению вершины с конфигурацией, содержащей заключительное состояние, выдаёт «да» и останавливается. Количество шагов, требуемых машине M_D для достижения такой вершины, не превосходит числа вершин в T', умноженного на константу — максимальное количество переходов из одной вершины в соседние. Число вершин в T' не больше суммы $1 + m + m^2 + \ldots + m^{t(n)} \leqslant (t(n)+1)m^{t(n)} \leqslant 2^{t(n)}m^{t(n)} = (2m)^{t(n)}$, где m — наибольшая степень исхода вершины в T' и m^{i-1} — верхняя граница числа вершин на i-м ярусе, $i = 1, 2, \ldots, t(n)+1$. Следовательно, временная сложность M_N есть $\mathrm{O}(c^{t(n)})$ для некоторой константы $c \geqslant 1$. Машина M_D достигает заключительного состояния для тех и только тех входных слов, для которых это делает M_N, поэтому $L(M_D) = L(M_N)$. ■

Неизвестно, однако, существует ли такая ДМТ M_D с полиномиальной временной сложностью, если временная сложность НМТ M_N полиномиальная.

Теорема 8.2. Для любой многоленточной НМТ M с временной сложностью $t(n)$ существует одноленточная НМТ M_1 с временной сложностью $\mathrm{O}(t^2(n))$, такая, что $L(M_1) = L(M)$.

Доказательство. См. также [2, лемма 10.1]. Для k-ленточной машины M лента в M_1 является $2k$-дорожечной и символ z в непустой клетке на ней представляет собой вектор $z = z_1 z_2 z_3 z_4 \ldots z_{2k-1} z_{2k}$, где z_{2j-1} есть символ в некоторой клетке на j-й ленте, а z_{2j} есть либо пустой символ, либо метка (скажем, «#»), свидетельствующая, что в M головка на j-й ленте обозревает именно эту клетку, $j = 1, 2, \ldots, k$. Машина M_1 моделирует работу машины M над заданным входным словом w по шагам — от 1-го до $t_M(n)$-го. Построим машину M_1, которая на моделирование одного шага M затрачивает не более $2t_M(n)$ своих шагов. В этом построении предполагается, что перед началом моделирования каждого шага M головка M_1 обозревает клетку на своей ленте, символ в которой содержит метку самой левой головки

машины M. Для начального шага это предположение обеспечивается надлежащей установкой, а для каждого последующего — процедурой моделирования текущего шага.

Моделирование t-го шага M в M_1 заключается в выполнении машиной M_1 параллельного перехода в дереве $T(M,w)$ одновременно из всех вершин t-го яруса в смежные им вершины $t+1$-го яруса для каждого одного значения t, $1 \leqslant t \leqslant t_M(n)$. Для любой одной вершины V на t-м ярусе данную операцию начинает один из клонов $M_1^{(i')}(t)$ машины M_1 при известном ему текущем состоянии q машины M, содержащемся в её конфигурации, коей является вершина V. Если $t=1$ и, следовательно, V есть корень $T(M,w)$, то $q=q_0$; в противном случае q определяется в результате выполнения этой операции над вершиной из предыдущего, $(t-1)$-го, яруса, смежной вершине V. Машина M_1 запоминает это q в своём текущем состоянии q'.

Работа машины $M_1^{(i')}(t)$ над вершиной V начинается с движения её головки на ленте вправо до тех пор, пока на дорожках с чётными номерами она не увидит каждую из всех k меток и не зафиксирует в своём состоянии набор $x = x_1x_2\ldots x_k$ символов, записанных на дорожках с номерами 1, 3, …, $2k-1$ в клетках, помеченных символом «#» на дорожках с номерами 2, 4, …, $2k$ соответственно. После этого машина $M_1^{(i')}(t)$ вычисляет значение $\delta(q,x)$ функции переходов машины M: $\delta(q,x) = \{(s^{(i)},(y_1^{(i)},d_1^{(i)}),(y_2^{(i)},d_2^{(i)}),\ldots,(y_k^{(i)},d_k^{(i)})) : i=1,2,\ldots,m\}$, а её клоны $M_1^{(i'_i)}(t)$, $i=1,\ldots,m$, параллельно моделируют все m вариантов перехода M, содержащихся в этом значении. Клон $M_1^{(i'_i)}(t)$ моделирует i-й из этих вариантов $(s^{(i)},y_1^{(i)}y_2^{(i)}\ldots y_k^{(i)},d_1^{(i)}d_2^{(i)}\ldots d_k^{(i)})$ следующим образом: переходит в некоторое состояние s', запоминая в нём $s^{(i)}$ как возможное состояние M на следующем $((t+1)$-м$)$ шаге, и сдвигает свою головку влево, пока не пройдёт через все k меток головок M, заменяя на $(2j-1)$-й дорожке символ x_j в отмеченной клетке символом $y_j^{(i)}$ и сдвигая её метку на $2j$-й дорожке влево или вправо или оставляя её на месте в соответствии со значением $d_j^{(i)}$, $j=1,2,\ldots,k$. После этого головка $M_1^{(i'_i)}(t)$ будет находиться правее самой левой метки головок в M не больше, чем

на две клетки. Её смещение на эти две или менее клеток влево завершает моделирование данного варианта перехода M.

В машине M_1 заключительное состояние q'_f достигается из начального состояния, если и только если (для тех и только тех входов, для которых) содержащееся в нём заключительное состояние q_f достигается из начального состояния в M. Это значит, что $L(M_1) = L(M)$. Кроме того, если M допускает слово w длины n не более, чем за $t(n)$ шагов, то M_1 делает это не более, чем за $2t^2(n)$ шагов, поскольку каждый шаг M моделируется в M_1 самое большее за $2t(n)$ шагов. Это значит, что временная сложность M_1 есть $\mathrm{O}(t^2(n))$. ■

9. Классы задач $\mathcal{P}$ и $\mathcal{NP}$

Формально говоря, задача (язык) L приналежит *классу* $\mathcal{P}$ или $\mathcal{NP}$, если существует соответственно ДМТ или НМТ M с полиномиальной временной сложностью, такая, что $L = L(M)$. Поскольку не исключено существование ДМТ M_D с экспоненциальной временной сложностью и НМТ M_N с полиномиальной такой сложностью, для которых $L(M_D) = L(M_N)$, нельзя утверждать справедливость включения $\mathcal{NP} \subseteq \mathcal{P}$ и вместе с ним равенства $\mathcal{P} = \mathcal{NP}$.

Примером нетривиальной задачи в $\mathcal{P}$ может служить поиск в графе с целочисленными весами рёбер остовного дерева (части графа со всеми вершинами и без циклов) с наименьшей суммой весов рёбер в ней. Для графа с m вершинами и e рёбрами задача решается в [2] на ДМТ за время $\mathrm{O}(n^4)$, где $n = m + e$. Примером задачи в $\mathcal{NP}$, но, предположительно, не в $\mathcal{P}$, служит задача коммивояжёра, требующая узнать, есть ли в том же графе гамильтонов цикл (проходящий через каждую вершину по одному разу) с суммой весов рёбер, не превосходящей заданного числа. Задача решается на НМТ за то же время [2].

10. NP-полные задачи

Алгоритм $A : \Sigma_1^* \to \Sigma_2^*$ с полиномиальной временной сложностью называется *полиномиальным сведением* задачи $L_1 \subset \Sigma_1^*$ к задаче $L_2 \subset \Sigma_2^*$, если для любого экземпляра $w_1 \in \Sigma_1^*$ имеет место $L_1(w_1) = L_2(A(w_1))$. В случае существования такого алгоритма говорят также, что L_1 *полиномиально сводима к* L_2.

Задача L из класса $\mathcal{NP}$ называется NP-*полной*, если для всякой задачи L' в $\mathcal{NP}$ существует полиномиальное сведение L' к L. В этом случае, если $L \in \mathcal{P}$ и $L' \in \mathcal{NP}$, то $L' \in \mathcal{P}$, ибо $L'(w') = L(A(w'))$, т. е. если некоторая NP-полная задача принадлежит $\mathcal{P}$, то $\mathcal{NP} \subseteq \mathcal{P}$ и, следовательно, $\mathcal{P} = \mathcal{NP}$. Таким образом, верна

Теорема 10.1. Все NP-полные задачи полиномиально сводимы одна к другой, и если $\mathcal{P} \neq \mathcal{NP}$, то все они принадлежат $\mathcal{NP} \setminus \mathcal{P}$.

Следующая теорема даёт метод построения новых NP-полных задач по уже имеющимся таким задачам.

Теорема 10.2. Если задача P_1 является NP-полной и существует полиномиальное сведение P_1 к P_2, то задача P_2 также NP-полна.

Доказательство. Если $P \in \mathcal{NP}$, $P(x) = P_1(A_1(x))$ и $P_1(x) = P_2(A(x))$, то $P(x) = P_2(A(A_1(x)))$. ■

11. NP-полнота задачи выполнимости

Рассматриваются булевы формулы, построенные из булевых (со значениями 0, 1) переменных множества $X = \{x_1, x_2, \ldots, x_m\}$ при помощи двухместных операций $\vee$ (дизъюнкция), $\wedge$ (конъюнкция) и одноместной операции $\neg$ (отрицание). Булева формула (БФ) называется *выполнимой*, если существует набор значений переменных в ней, при котором она обращается в 1. Задача выполнимости состоит в том, что требуется выяснить, выполнима ли заданная БФ φ.

Имея в виду, к примеру, индуктивное определение булевой формулы, можно сказать, что любая БФ есть слово в алфавите $\Sigma = \{x_1, \ldots, x_m, \vee, \wedge, \neg, (,)\}$. Представляя индекс i переменной x_i его двоичным кодом некоторой длины, можно записать любую БФ в виде слова в восьмизначном алфавите $\Sigma = \{x, 0, 1, \vee, \wedge, \neg, (,)\}$. Будем называть это слово *кодом* данной БФ. Длина k кода формулы и длина n самой формулы находятся в отношении $k \leqslant n \log m$ и разница между ними ограничена полиномом от n: $k - n \leqslant n \log m - n < n \log m$. Поэтому, интересуясь решением задач о БФ с полиномиальной сложностью от длины входа, обычно не делают различий между длинами булевой формулы и её кода, между булевыми формулами и их кодами,

между алфавитами для записи последних как слов. Рассматривая задачу выполнимости БФ, мы тоже будем придерживаться этого правила.

Обозначим W множество всех слов в алфавите Σ, которые представляют собой всевозможные БФ с переменными в X. Пусть также SAT есть язык в этом алфавите, состоящий из всех слов в W, которые представляют собой всевозможные выполнимые БФ. Тогда задачу выполнимомости можно сформулировать как задачу принадлежности слова из W языку SAT и, в соответствии с принятым соглашением, называть SAT-задачей.

Теорема 11.1 (S. C. Cook). Задача SAT NP-полна.

Доказательство. Сначала докажем, что SAT$\in \mathcal{NP}$, т. е. что SAT-задача решается недетерминированной МТ за полиномиальное время. Для этого возьмём НМТ M, которая для БФ φ, заданной на её ленте словом в алфавите Σ, и каждого $i \in \{1, 2, \ldots, m\}$ параллельно вычисляет БФ $\varphi(\alpha a)$ для всех $\alpha \in \{0,1\}^{i-1}$ и $a \in \{0,1\}$, где индуктивно $\varphi(\alpha a)$ определяется подстановкой в $\varphi(\alpha)$ константы a вместо переменной x_i и $\varphi(\Lambda) = \varphi$. Если для некоторого $\alpha \in \{0,1\}^m$ оказывается $\varphi(\alpha) = 1$, то выдаётся ответ «φ выполнима», в противном случае — «φ невыполнима». Время работы МТ M над БФ длиной n есть $\mathrm{O}(n^c)$, поскольку всё это время тратится на выполнение m раз операции подстановки константы вместо некоторой переменной одновременно во все имеющиеся на этот момент БФ и всякий раз эта операция выполняется за полиномиальное время от величины не больше n.

Теперь докажем, что к SAT полиномиально сводится любой язык L из $\mathcal{NP}$. По определению последнего, $L \in \mathcal{NP}$ тогда и только тогда, когда существует НМТ M с полиномиальной сложностью $t(n)$, решающая L, т. е. такая, что $L(M) = L$. Без ограничения общности ввиду теоремы 8.2 предполагаем, что НМТ M 1-ленточная. По определению $L(M)$, равенство $L = L(M)$ имеет место, если и только если для любого слова $w \in \Sigma^*$ длины n в алфавите Σ языка L в M найдётся последовательность конфигураций (путь в дереве $T(M, w)$) $\alpha = \alpha_0\alpha_1 \ldots \alpha_l$, где $\alpha_0 = q_0 w$ — начальная конфигурация, α_l содержит в себе заключительное состояние M, $l \leqslant t(n)$ и ни одна конфигурация не занимает на ленте более $t(n)$ клеток. Далее такая последовательность называется *допускающей* для w в M. По определению полиномиальной сво-

димости, язык L полиномиально сводится к SAT, если и только если существует полиномиальный алгоритм A, преобразующий любое слово $w \in \Sigma^n$ в некоторую булеву формулу $E(M, w)$ полиномиальной длины от n, которая принадлежит SAT тогда и только тогда, когда $w \in L$. Условие $E(M, w) \in SAT$, как видим, равносильно существованию в M допускающей последовательности α для w, и для доказательства полиномиальной сводимости к SAT языка L достаточно построить $E(M, w)$, которая будет выполнимой, если и только если α удовлетворяет всем условиям на допускающую последовательность конфигураций для w в M.

Без ограничения общности будем считать далее для простоты, что $l = t(n)$ (этого можно достичь, вводя тождественные переходы) и длина любой конфигурации в α также равна $t(n)$ (этого можно достичь за счёт пустых символов). Пусть также $X = \{x_1, x_2, \ldots, x_m\}$ есть ленточный алфавит, $x_1 = \Lambda$, $Q = \{q_0, q_1, \ldots, q_{r-1}\}$ — множество состояний, $q_f \in Q$ — заключительное состояние в M, $w = x_{a_1} x_{a_2} \ldots x_{a_n}$.

Опишем формулой $E(M, w)$ конфигурации α_k и переходы между ними в M. Для этого введём следующие булевы переменные в количестве $\mathrm{O}(t^2(n))$:

— $y_{ijk} = 1 \Leftrightarrow z_i = x_j$ в α_k, $1 \leqslant i \leqslant t(n)$, $1 \leqslant j \leqslant m$ и $0 \leqslant k \leqslant t(n)$ (здесь и далее z_i обозначает ленточный символ из конфигурации α_k в i-й клетке ленты);
— $u_{sk} = 1 \Leftrightarrow$ в α_k МТ M находится в состоянии q_s;
— $h_{ik} = 1 \Leftrightarrow$ в α_k головка обозревает i-ю клетку (z_i).

Дальше нам понадобится булева функция $g(b_1, b_2, \ldots, b_v) = (b_1 \vee \vee b_2 \vee \ldots \vee b_v) \wedge \bigwedge\limits_{i \neq j} (\neg b_i \vee \neg b_j)$, равная 1 тогда и только тогда, когда ровно одна переменная равна 1. В её выражении $\mathrm{O}(v^2)$ символов (v множителей, в каждом не более v слагаемых).

Утверждение о том, что последовательность слов $\alpha_0 \alpha_1 \ldots \alpha_{t(n)}$ в алфавите $X \cup Q$ на ленте МТ является допускающей для w последовательностью конфигураций в M, равносильно совокупности следующих предложений и истинности сопоставленных им подформул в $E(M, w)$:

1) в каждом слове в α головка обозревает ровно одну клетку — истинна формула $A = A_0 \wedge A_1 \wedge \ldots \wedge A_{t(n)}$, где $A_k = g(h_{1k}, h_{2k}, \ldots, h_{t(n)k})$; её длина равна $\mathrm{O}(t^3(n))$, в ней $t(n)$ множителей, в каждом не более $t^2(n)$ вхождений переменных;

2) в каждом слове в α в каждой клетке ленты ровно один символ — истинна формула $B = \bigwedge\limits_{i,k} B_{ik}$, где $B_{ik} = g(y_{i1k}, y_{i2k}, \ldots, y_{imk})$; её длина — $\mathrm{O}(t^2(n))$, в ней: длина каждой B_{ik} не зависит от n (константа); i — номер клетки, $i = 1, 2, \ldots, t(n)$; k — номер слова α_k, $k = 0, 1, \ldots, t(n)$; $t^2(n)$ множителей-констант;

3) каждое слово в α содержит ровно одно состояние — истинна формула $C = \bigwedge\limits_{0 \leqslant k \leqslant t(n)} g(u_{0k}, u_{1k}, \ldots, u_{r-1,k})$; её длина — $\mathrm{O}(t(n))$, в ней $t(n)$ множителей, каждый множитель не зависит от n (константа);

4) при переходе от одного слова в α к следующему может измениться только содержимое обозреваемой клетки — истинна формула $D = \bigwedge\limits_{i,j,k} (h_{ik} \vee (y_{ijk} \sim y_{ijk+1}))$; её длина после раскрытия эквиваленции $\sim$ равна $\mathrm{O}(t^2(n))$; формула под знаком $\bigwedge$ означает, что в α_k головка обозревает клетку i либо содержимое клетки i не меняется, что равносильно любому из следующих двух высказываний: а) в α_k головка обозревает клетку i и содержимое клетки i меняется, либо содержимое клетки i не меняется, б) в α_k головка обозревает клетку i, либо в α_k головка не обозревает клетку i и содержимое клетки i не меняется;

5) при переходе от одного слова к следующему состояние машины, положение головки и содержимое клетки изменяются в соответствии со значением функции переходов δ управляющего автомата в M — истинна формула $E = \bigwedge\limits_{i,j,s,k} E_{ijsk}$, где $E_{ijsk} = \neg y_{ijk} \vee \neg h_{ik} \vee \neg u_{sk} \vee \bigvee\limits_{l} (y_{ij_l,k+1} \wedge u_{s_l,k+1} \wedge h_{i_l,k+1})$, l пробегает все варианты перехода, когда M обозревает в i-й клетке символ x_j в состоянии q_s. Иначе говоря, для каждого варианта перехода $(q, y, d) \in \delta(q_s, x_j)$ существует одно значение l, для которого $x_{j_l} = y$, $q_{s_l} = q$ и число i_l в зависимости от d есть $i - 1$, $i + 1$ или i, а именно: $d = L$, $i_l = i - 1$, или $d = R$, $i_l = i + 1$, или $d = S$, $i_l = i$ соответственно. Формула E_{ijsk} означает одно из следующих четырёх утверждений: в α_k клетка i не содержит символа x_j, головка не обозревает клетку i, машина не находится в состоянии q_s, очередная конфигурация получается из предыдущей в соответствии со значением функции переходов. Число

вариантов (q, y, d) конечное и E_{ijsk} имеет ограниченную длину, не зависящую от n, поэтому E имеет длину $O(t^2(n))$, ибо i и k пробегают по $t(n)$ значений, s и j — по конечному (не зависящему от n) числу значений;

6) первое слово в α является начальной конфигурацией — истинна формула $F = u_{00} \wedge h_{10} \wedge \bigwedge\limits_{1 \leqslant i \leqslant n} y_{ia_i 0} \wedge \bigwedge\limits_{n < i \leqslant t(n)} y_{i10}$ длины $\mathrm{O}(t(n))$;

7) состояние в последнем слове заключительное — истинна формула $G = u_{ft(n)}$.

Следовательно, $E(M, w) = A \wedge B \wedge C \wedge D \wedge E \wedge F \wedge G$ и временная сложность $E(M, w)$ есть $\mathrm{O}(t^3(n))$. ■

Следствие 2. Задача выполнимости КНФ NP-полна.

Доказательство. В $E(M, w)$ формулы A, B, C, F, G представлены в КНФ, формула D приводится к КНФ по правилу $(a \sim b) \vee c = ab \vee (\neg a)(\neg b) \vee c = (a \vee \neg b \vee c)(\neg a \vee b \vee c)$ с увеличением длины не более, чем в 2 раза, длина формулы E_{ijsk} в E не зависит от n и после приведения её к КНФ по правилу $a \vee bc = (a \vee b)(a \vee c)$ удлиняется в константу раз. ■

12. Другие NP-полные задачи

Следующие задачи NP-полны.

Задача 3-выполнимости (3 SAT). Требуется узнать, выполнима ли КНФ, в которой каждая элементарная дизъюнкция имеет ранг 3 (состоит из трёх литералов). NP-полнота этой задачи доказывается полиномиальным сведением к ней задачи SAT КНФ.

Задача независимого множества (НМ). Подмножество вершин в графе называется независимым, если в нём нет смежных вершин. Требуется выяснить, есть ли в графе независимое множество заданной мощности. NP-полнота задачи доказывается полиномиальным сведением к ней задачи 3 SAT.

Задача вершинного покрытия (ВП). Подмножество V вершин в графе называется вершинным покрытием графа, если любое ребро графа инцидентно хотя бы одной вершине в V. Требуется выяснить, есть ли в графе вершинное покрытие заданной мощности. NP-полнота задачи доказывается полиномиальным сведением к ней задачи НМ.

Задача о клике. Подмножество вершин в графе называется *кликой*, если любые две вершины в нём смежные. Задача: есть ли в графе клика заданной мощности? Её NP-полнота доказывается полиномиальным сведением к ней задачи ВП.

Задача изоморфизма: есть ли в графе подграф, изоморфный другому графу. Её NP-полнота доказывается полиномиальным сведением к ней задачи о клике.

Задача раскраски: является ли граф k-раскрашиваемым, т. е. можно ли каждую вершину графа окрасить в один из k цветов, так, что никакие две смежные вершины не будут окрашены в один цвет? NP-полнота задачи доказывается полиномиальным сведением к ней задачи 3SAT.

Задача ориентированного гамильтонова цикла (ОГЦ). Требуется узнать, есть ли в ориентированном графе гамильтонов цикл (проходит через каждую вершину ровно один раз). NP-полнота задачи доказывается полиномиальным сведением к ней задачи 3 SAT.

Задача неориентированного гамильтонова цикла (ГЦ). Требуется узнать, есть ли в неориентированном графе гамильтонов цикл. NP-полнота задачи доказывается сведением к ней задачи ОГЦ.

Задача коммивояжёра (КОМ). Требуется узнать, есть ли в неориентированном графе с целочисленными весами рёбер гамильтонов цикл с заданной суммой весов рёбер. NP-полнота задачи доказывается сведением к ней задачи ГЦ.

Примерами других (неграфовых) NP-полных задач являются: задача целочисленного линейного программирования о совместности системы линейных неравенств с целочисленными константами и переменными; задача о возможности разбиения множества на блоки, принадлежащие заданному покрытию этого множества; задача о возможности разбиения системы чисел на две части с одинаковыми суммами чисел в частях; задача нахождения решения системы нелинейных (достаточно — квадратных) уравнений над конечным полем и многие другие.

В действительности известны тысячи NP-полных задач, и ни для одной из них не найдено полиномиального алгоритма решения. Этот факт говорит в пользу неравенства $\mathcal{P} \neq \mathcal{NP}$ и тем самым укрепляет убеждение в справедливости гипотезы о том,

что все NP-полные задачи требуют для своего решения экспоненциального времени, т. е. являются труднорешаемыми.

13. Генерическая сложность

13.1. Генерическая разрешимость

Под генерической сложностью задачи понимается её сложность в типичном случае, т. е. сложность алгоритма её решения на почти всех входных данных [3]. Это понятие обеспечивает единую основу для изучения и сравнения вычислительной сложности как решаемых, так и нерешаемых задач, что невозможно в рамках понятия сложности в худшем случае или в среднем. Оказалось, например, что задача останова машины Тьюринга (МТ), хорошо известная как нерешаемая, легко решается на большинстве входных данных — на так называемых генерических множествах МТ. Показано также, что среди нерешаемых задач на группах и полугруппах есть такие, которые тоже легко решаются на некоторых генерических множествах входов. Разумеется, есть и задачи, так называемые «абсолютно» нерешаемые, которые нерешаемы на любом генерическом множестве данных. О некоторых из них мы ещё поговорим в этой главе.

Пусть I есть некоторое множество и $L \subseteq I$. Обозначим $D = (L, I)$ задачу разрешения, в которой требуется для любого $x \in I$ решить, принадлежит ли x к L или нет. Мы называем I *множеством входов* задачи D и предполагаем, что оно снабжено *функцией размера* $s : I \to \mathbb{N}$. Обычно под размером $s(w)$ входа $w \in I$ понимается длина его описания. Мы предполагаем также, что для любого натурального $n \in \mathbb{N}$ подмножество $I_n = \{w \in I : s(w) = n\}$ всех элементов в I, имеющих размер n, конечное. Из этого следует, в частности, что существует вычислимая биекция между I и $\mathbb{N}$, позволяющая элементы в I нумеровать натуральными числами и осуществлять их «последовательный перебор». Для подмножества $S \subseteq I$ и $n \in \mathbb{N}$ определим $\rho_n(S) = \dfrac{|S \cap I_n|}{|I_n|}$. Это есть вероятность попадания в S элемента из I_n при его случайной и равновероятной выборке в I_n. Её предел $\rho(S) = \lim\limits_{n\to\infty} \rho_n(S)$, если он существует, называется *асимптотической плотностью* S. Подмножество S называется *генерическим*, если $\rho(S) = 1$, и *пренебрежимым*, если $\rho(S) = 0$.

Понятно, что S генерическое, если и только если его дополнение $I \setminus S$ пренебрежимое.

Стоит заметить, что асимптотическая плотность позволяет не только различать между собой «большие» (генерические) и «маленькие» (пренебрежимые) множества, но и давать средство для различения между разными большими (или маленькими) множествами. Так, подмножество $S \subseteq I$ называется *непренебрежимым*, если $\rho(S) > 0$. Говорят также, что $\rho_n(S)$ имеет *полиномиальную сходимость* (к $\rho(S)$), если $|\rho(S) - \rho_n(S)| = \mathrm{o}(n^{-c})$ для некоторой натуральной константы c. В этом случае генерическое подмножество S называется *сильно генерическим*. Подмножество S называется *сильно пренебрежимым*, если его дополнение $I \setminus S$ сильно генерическое. Можно также сказать, что подмножество S *сильно пренебрежимое*, если существуют такие константы σ и c, что $0 < \sigma < 1$, $c > 0$ и $\dfrac{|S \cap I_n|}{|I_n|} < c\sigma^n$, т. е. последовательность долей в S всех входов размера n экспоненциально быстро сходится к 0, и тогда S *сильно генерическое*, если $I \setminus S$ сильно пренебрежимое. Заметим также, что подмножество пренебрежимого или сильно пренебрежимого множества является пренебрежимым или сильно пренебрежимым соответственно.

Аналогичным образом (с использованием $\mathrm{o}(c^{-n})$ вместо $\mathrm{o}(n^{-c})$) определяются *экспоненциальная сходимость* и *экспоненциально генерические* подмножества.

Напомним, что задача $D = (L, I)$ называется *алгоритмически разрешимой*, если существует алгоритм $\mathcal{A} : I \to \{0, 1\}$, такой, что для любого $x \in I$ если $x \in L$, то $\mathcal{A}(x) = 1$, а иначе $\mathcal{A}(x) = 0$; в этом случае $\mathcal{A}$ называется *алгоритмом разрешения*, или *разрешающим алгоритмом* задачи D. Для любого подмножества $S \subseteq I$ задача разрешения $D_S = (S \cap L, S)$ называется *частью* или *ограничением* задачи D. Её алгоритм разрешения, если он существует, называется *частичным алгоритмом разрешения* для задачи D на подмножестве S. Его множество останова, как видно, совпадает с S. В случае существования такого алгоритма задача D называется *алгоритмически разрешимой на подмножестве S*.

Задача разрешения D, алгоритмически разрешимая на некотором генерическом, сильно генерическом или непренебрежимом подмножестве входов, называется соответственно *генериче-*

ски, сильно генерически или *абсолютно генерически разрешимой*. Другими словами, задача D генерически, сильно генерически или абсолютно генерически разрешима, если для неё существует частичный алгоритм разрешения $\mathcal{A}$ (на каком-либо подмножестве входов), обладающий соответственно генерическим, сильно генерическим или непренебрежимым множеством останова $H_{\mathcal{A}}$.

В согласии с данным определением, задача разрешения D считается *генерически, сильно генерически* или *абсолютно* (*генерически*) *неразрешимой*, если она алгоритмически неразрешима ни на одном соответственно генерическом, сильно генерическом или непренебрежимом подмножестве входов. Иначе говоря, задача D генерически, сильно генерически или абсолютно (генерически) неразрешима, если для неё не существует частичного алгоритма разрешения $\mathcal{A}$ (на каком-либо подмножестве входов), имеющего соответственно генерическое, сильно генерическое или непренебрежимое множество останова $H_{\mathcal{A}}$.

Всякий частичный разрешающий алгоритм $\mathcal{A}$ для задачи D на каком-либо генерическом (сильно генерическом или непренебрежимом) подмножестве S её входов называется *генерическим* (*сильно генерическим* или *абсолютно генерическим*) *алгоритмом* решения этой задачи. Его множество останова $H_{\mathcal{A}}$ совпадает с S и является, таким образом, генерическим (сильно генерическим или непренебрежимым). Обратное может не выполняться, а именно: алгоритм с генерическим любого типа множеством останова $H_{\mathcal{A}} = S \subseteq I$ может и не быть разрешающим алгоритмом для D на S. Вместе с тем задача D генерически (сильно генерически или абсолютно генерически) разрешима, если и только если для неё существует частичный алгоритм разрешения с генерическим (сильно генерическим или непренебрежимым) множеством останова.

13.2. Генерическая разрешимость задачи останова ДМТ

Эта задача является важнейшей из алгоритмически неразрешимых задач. Покажем её генерическую разрешимость [3, 4]. Будем рассматривать множество P всех одноленточных ДМТ, где произвольная ДМТ M имеет конечное множество состояний Q, начальное состояние q_0, состояние остановки q_f не из Q и функцию переходов $\delta : Q \times \{0,1\} \to (Q \cup \{q_f\}) \times \{0,1\} \times \{L,R\}$. В ней

для $q, s \in Q$, $a, b \in \{0, 1\}$ и $d \in \{L, R\}$ *команда* $\delta(q, a) = (s, b, d)$ означает, что если машина находится в состоянии q и её головка обозревает символ a, то она должна перейти в состояние s, записать на ленту символ b и переместить головку на одну клетку влево, если $d = L$, или вправо, если $d = R$. Работа M происходит путём итеративного исполнения её команд до достижения финального состояния q_f, в котором она останавливается. Если машина работает на односторонней ленте и пытается сдвинуть головку левее самой левой клетки ленты, то считается, что машина достигает конфигурации прерывания и её работа прекращается.

Далее число состояний в МТ называется её *размером*, множество всех МТ в P размера n обозначается P_n. Поскольку области определения и значений функции переходов МТ размера n имеют мощности $2n$ и $4(n + 1)$ соответственно, количество всех МТ в P_n равно $|P_n| = (4(n + 1))^{2n}$.

Естественно за меру размера подмножества МТ в P принять его асимптотическую плотность и считать подмножество $S \subseteq P$ генерическим, если $\rho(S) = 1$, где $\rho(S) = \lim\limits_{n\to\infty} \dfrac{|S \cap P_n|}{|P_n|}$, если этот предел существует.

Задача останова для МТ формулируется как задача разрешения (H, P) для подмножества H всех МТ в P, которые, начиная работу в состоянии q_0 над пустой лентой (заполненной символами 0), через конечное число шагов останавливаются, достигая заключительного состояния q_f или конфигурации прерывания. Задача заключается в выяснении, является ли множество H генерически разрешимым, т. е. существует ли генерический алгоритм, который по функции переходов МТ на его входе узнаёт, принадлежит ли эта МТ множеству H. Следующий результат свидетельствует о положительном ответе этой задачи. Он получен в [4] и справедлив для МТ с односторонней и изначально пустой лентой.

Теорема 13.1. В множестве P существует подмножество S со следующими свойствами:

1) S генерическое;
2) S разрешимо за полиномиальное время;
3) ограничение $(H \cap S, S)$ задачи останова (H, P) на S полиномиально разрешимо.

Следствие 3. Задача останова (H, P) генерически разрешима.

Доказательство. Свойство 3 является утверждением существования алгоритма разрешения полиномиальной сложности для ограничения $(H \cap S, S)$ задачи (H, P) на генерическом подмножестве S. ■

Фактически же свойства 2 и 3 подмножества S в теореме говорят о большем, а именно о том, что существует алгоритм, который для любой машины M в P, если она принадлежит S, позволяет сначала убедиться в этом за полиномиальное время, а затем опять же за полиномиальное время проверить, принадлежит ли это M множеству $H \cap S$, и в случае положительного или отрицательного результата проверки принять $M \in H$ или $M \notin H$ соответственно.

Вместе с тем в [5] доказано, что задача останова МТ в её классической постановке сильно генерически неразрешима, т. е. неразрешима на любом сильно генерическом подмножестве входов. Для доказательства этого, а также существования абсолютно неразрешимых задач нам понадобятся далее понятие перечислимого множества и некоторые свойства перечислимости.

13.3. Перечислимость

Подмножество $S \subseteq I$ называется *перечислимым* (enumerable, computably enumerable или recursively enumerable), если оно является множеством всех членов в некоторой вычислимой последовательности элементов из I, или, что то же самое, существует алгоритм $\mathcal{A} : \mathbb{N} \to \mathcal{S}$ со свойством: $S = \{\mathcal{A}(n) : n \in \mathbb{N}\}$. В этом случае говорят, что S *перечисляется алгоритмом* $\mathcal{A}$.

По определению, множество натуральных чисел перечислимое. Множества целых и рациональных чисел также перечислимы. Перечислимо и подмножество перечислимого множества. Множество всех перечислимых множеств замкнуто относительно операций объединения, пересечения, декартова произведения. Дополнение перечислимого подмножества может не быть перечислимым. Множество всех конечных последовательностей с членами из перечислимого множества перечислимо. Множество всех полиномов от одного переменного с рациональными коэффициентами перечислимо. Не являются перечислимыми множе-

ство действительных чисел в интервале (0, 1) и множество всех действительных чисел. Расширение неперечислимого множества также неперечислимо.

Существуют перечислимые неразрешимые подмножества. Имеет место следующая теорема Поста: подмножество S разрешимо, если и только если S и $I \setminus S$ перечислимы.

Справедливо также следующее утверждение: подмножество S перечислимо, если и только если оно есть множество останова некоторого алгоритма. В самом деле, если S перечисляется некоторым алгоритмом $\mathcal{A}$, то $S = H_{\mathcal{B}}$ для алгоритма $\mathcal{B}$, который, получив на вход $x \in I$, последовательно перебирает натуральные числа, пока не найдётся такое n в $\mathbb{N}$, что $\mathcal{A}(n) = x$, и останавливается. Обратно, если $S = H_{\mathcal{B}}$ для некоторого алгоритма $\mathcal{B}$, то S перечисляется алгоритмом $\mathcal{A}$, который для заданного на его входе натурального n последовательно перебирает элементы в I, пока не найдётся такой x в I, что алгоритм $\mathcal{B}$ останавливается через n шагов после запуска на входе x, и алгоритм $\mathcal{A}$ выдаёт на выходе $x = \mathcal{A}(n)$.

13.4. Сильная генерическая неразрешимость задачи останова ДМТ

Пусть $\varepsilon(M)$ обозначает код функции переходов ДМТ $M \in P$ в алфавите $\{1\}$. Классическая проблема останова формулируется как задача разрешения (HP, P), где HP есть подмножество тех МТ M в P, которые останавливаются на своих кодах $\varepsilon(M)$, т. е. чьё множество останова H_M состоит из одного слова — $\varepsilon(M)$. Здесь мы убедимся в её сильной генерической неразрешимости, т. е. в несуществовании частичного алгоритма разрешения для (HP, P), имеющего сильно генерическое множество останова.

Для заданного натурального n и произвольной МТ M с $k \leqslant n$ состояниями обозначим $C(M)$ множество всех таких её расширений M^* с n состояниями, в которых новые $n - k$ состояний недостижимы из состояний в M. Ясно, что $C(M) \subseteq P_n$ и на состояниях в M поведение любой МТ M^* в $C(M)$ совпадает с поведением M.

Лемма 13.1. Множество $C(M)$ не сильно пренебрежимое.

Доказательство. Имеем $|P_n| = (4(n+1))^{2n}$, $|C(M)| = (4(n+1))^{2(n-k)}$, поэтому $\dfrac{|C(M) \cap P_n|}{|P_n|} = \dfrac{1}{(4(n+1))^{2k}}$, откуда

$\rho(C(M)) = 0$, множество $C(M)$ пренебрежимое и, следовательно, множество $P \setminus C(M)$ генерическое, т. е. $\rho(P \setminus C(M)) = 1$, ввиду чего

$$\left|\rho(P \setminus C(M)) - \frac{|(P \setminus C(M)) \cap P_n|}{|P_n|}\right| = \left|\frac{|P_n| - |(P \setminus C(M)) \cap P_n|}{|P_n|}\right| =$$
$$= \frac{|C(M) \cap P_n|}{|P_n|} = \frac{1}{(4(n+1))^{2k}},$$

что не есть $o(n^{-c})$ ни для какой положительной константы c, означая, что множество $P \setminus C(M)$ не сильно генерическое, а, значит, его дополнение $C(M)$ не сильно пренебрежимое. ■

Теорема 13.2. Задача останова (HP, P) МТ в P на её коде сильно генерически неразрешима.

Доказательство. Предположим противное: существует частичный алгоритм $\mathcal{A}$ разрешения для задачи (HP, P), имеющий сильно генерическое множество останова $H_{\mathcal{A}}$. По определению, это есть разрешающий алгоритм для $(HP \cap H_{\mathcal{A}}, H_{\mathcal{A}})$, такой, что для любой МТ $M \in H_{\mathcal{A}}$ если $M \in HP \cap H_{\mathcal{A}}$, то $\mathcal{A}(M) = 1$, иначе $\mathcal{A}(M) = 0$. Построим на $H_{\mathcal{A}}$ алгоритм $\mathcal{B}$, положив $\mathcal{B}(M) = 1$, если $\mathcal{A}(M) = 0$, и $\mathcal{B}(M) = -$ (неопределено), если $\mathcal{A}(M) = 1$. По построению, $H_{\mathcal{B}} = H_{\mathcal{A}} \setminus HP$, поэтому множество $H_{\mathcal{A}} \setminus HP$ перечислимо и существует МТ M с множеством останова $H_M = \{\varepsilon(M') : M' \in H_{\mathcal{A}} \setminus HP\}$. Сильно генерическое $H_{\mathcal{A}}$ является дополнением в P множества $P \setminus H_{\mathcal{A}}$, поэтому последнее сильно пренебрежимое, а множество $C(M)$ по лемме 13.1 таковым не является. Следовательно, $C(M) \not\subseteq P \setminus H_{\mathcal{A}}$ и $C(M) \cap H_{\mathcal{A}} \neq \varnothing$. Пусть $M^* \in C(M) \cap H_{\mathcal{A}}$. Тогда для любой МТ $M' \in P$ можно записать $\varepsilon(M') \in H_{M^*} \Leftrightarrow \varepsilon(M') \in H_M \Leftrightarrow M' \in H_{\mathcal{A}} \setminus HP$. В частности, $\varepsilon(M^*) \in H_{M^*} \Leftrightarrow M^* \in H_{\mathcal{A}} \setminus HP \Rightarrow (M^* \notin HP) \Leftrightarrow$ $\Leftrightarrow (\varepsilon(M^*) \notin H_{M^*})$. Здесь последняя эквиваленция верна по определению HP. Полученное противоречие доказывает теорему. ■

13.5. Существование абсолютно неразрешимых задач

Здесь мы рассматриваем задачи, которые неразрешимы на любом непренебрежимом подмножестве [3]. Пусть $I = \{0,1\}^*$, $I_n = \{0,1\}^n$, $S \subseteq I$ и $\rho(S)$ — асимптотическая плотность подмножества S. Говорят, что задача принадлежности для S в I

абсолютно неразрешима, если не существует алгоритма $\mathcal{A}$, решающего эту задачу и имеющего непренебрежимое множество останова $H_{\mathcal{A}}$ (с положительной асимптотической плотностью $\rho(H_{\mathcal{A}}) > 0$).

Подмножество $S \subseteq I$ называется *иммунным*, если оно бесконечное и не содержит в себе бесконечных перечислимых подмножеств множества I.

Лемма 13.2. В I существует иммунное генерическое подмножество.

Доказательство. Пусть $S_1, S_2, \ldots$ есть последовательность всех бесконечных перечислимых подмножеств в I. Ввиду конечности всякого множества I_n и бесконечности всякого множества S_n, $n = 1, 2, \ldots$, существует последовательность слов $w_1, w_2, \ldots$ в I, такая, что $w_n \in S_n$ и $|w_n| < |w_{n+1}|$ для всех n (иначе S_n будет конечным). Пусть $R = I \setminus \{w_1, w_2, \ldots\}$. Множество R иммунное, так как $w_n \in S_n \setminus R$ и $\neg(S_n \subseteq R)$ для всех n. Множество R генерическое, так как $I_n \setminus R \subseteq I \setminus R = \{w_1, w_2, \ldots\}$, $|I_n \setminus R| \leqslant 1$, $|R \cap I_n| \geqslant 2^n - 1$, $\dfrac{|R \cap I_n|}{|I_n|} \geqslant \dfrac{2^n - 1}{2^n}$ и $\rho(R) = 1$. ■

Теорема 13.3. Для каждого генерического иммунного подмножества R в I задача разрешения (R, I) абсолютно неразрешима.

Доказательство. Предположим противное, а именно: пусть $\mathcal{A}$ есть алгоритм, решающий задачу (R, I), $H_{\mathcal{A}}$ — его непренебрежимое множество останова и $\rho(H_{\mathcal{A}}) > 0$. Положим $\overline{R} = I \setminus R$. Имеем:

$$\rho(R) = \lim_{n\to\infty} \frac{|R \cap I_n|}{|I_n|} = 1, \quad \rho(\overline{R}) = \lim_{n\to\infty} \frac{|\overline{R} \cap I_n|}{|I_n|} = 0,$$

$$\rho(H_{\mathcal{A}}) = \lim_{n\to\infty} \frac{|H_{\mathcal{A}} \cap I_n|}{|I_n|} =$$

$$= \lim_{n\to\infty} \left(\frac{|R \cap H_{\mathcal{A}} \cap I_n|}{|I_n|} + \frac{|\overline{R} \cap H_{\mathcal{A}} \cap I_n|}{|I_n|} \right) = \lim_{n\to\infty} \frac{|R \cap H_{\mathcal{A}} \cap I_n|}{|I_n|} =$$

$$= \rho(R \cap H_{\mathcal{A}}).$$

Следовательно, $\rho(R \cap H_{\mathcal{A}}) > 0$; в частности, $R \cap H_{\mathcal{A}}$ — бесконечное множество (иначе $\rho(R \cap H_{\mathcal{A}}) = 0$). Заметим также, что $R \cap H_{\mathcal{A}}$ —

перечислимое подмножество в I, так как $w \in R \cap H_{\mathcal{A}}$, если и только если, по определению $H_{\mathcal{A}}$ и R, алгоритм $\mathcal{A}$ останавливается на входном слове w и выдаёт 1, что означает допускаемость множества $R \cap H_{\mathcal{A}}$ алгоритмом $\mathcal{A}$. Таким образом, $R \cap H_{\mathcal{A}}$ есть бесконечное перечислимое подмножество в I, содержащееся в иммунном множестве R, что противоречит условию иммунности последнего. Это противоречие и доказывает теорему. ■

13.6. Генерическая сложность дискретного логарифмирования

Покажем (по [6]), что задача дискретного логарифма генерически трудноразрешима, если она такова в худшем случае (для всех входов). Напомним постановку задачи: для заданных простого числа p, порождающего элемента g в поле $\mathrm{GF}(p)$ и ненулевого элемента $a \in \mathrm{GF}(p)$ найти число $\log_g a = x \in \mathrm{GF}(p)$, для которого $g^x = a$.

Пусть далее $P = \{p_1, p_2, \ldots, p_n, \ldots\}$ есть последовательность всех простых чисел, удовлетворяющих условию $2^n \leqslant p_n < 2^{n+1}$ для всех n — так называемая экспоненциальная последовательность. Будем рассматривать поставленную задачу дискретного логарифма в случае, когда множество I её входов состоит из всех троек (p, g, a) с p в P, а подмножество I_n её входов размера n — из троек (p, g, a) с фиксированными p, g и произвольным $a \in \{1, \ldots, p-1\}$, где $2^n \leqslant p < 2^{n+1}$. Под алгоритмом её решения будем понимать всякий алгоритм с множеством входов I, который останавливается на всех его входах с выдачей для каждого входа (p, g, a) либо значения $x = \log_g a$ из $\mathrm{GF}(p)$, либо отказа — сообщения о невозможности вычислить это значение. Алгоритм $\mathcal{A}$ называется *генерическим*, если множество $S_{\mathcal{A}}$ входов в I, на которых он останавливается с выдачей отказа, является пренебрежимым, т. е. если вероятность $\rho_n(S_{\mathcal{A}}) = \dfrac{|S_{\mathcal{A}} \cap I_n|}{|I_n|}$ стремится к 0 с ростом n. Алгоритм называется *вероятностным*, если для случайно и равномерно выбранного входа в I он останавливается с выдачей отказа с вероятностью меньше $1/2$.

Теорема 13.4. Пусть $p = p_n \in P$ и n достаточно большое. Если существует генерический алгоритм $\mathcal{A}$ полиномиальной сложности, вычисляющий дискретный логарифм в $\mathrm{GF}(p)$ для типичных (не в $S_{\mathcal{A}}$) входов в I_n, то существует вероятностный алго-

ритм $\mathcal{B}$ полиномиальной сложности, вычисляющий дискретный логарифм для всех входов в I_n.

Доказательство. Требуемый алгоритм $\mathcal{B}$ на входе $(p, g, a) \in \in I_n$ выполняет следующие действия:

1) генерирует случайно и равномерно $y \in \{0, 1, \ldots, p-1\}$, вычисляет $a' = ag^y$ и применяет алгоритм $\mathcal{A}$ к входу (p, g, a');
2) если $\mathcal{A}(p, g, a') = x' = \log_g a'$, то

$$\log_g a = x = (x' - y) \bmod (p - 1),$$

ибо $ag^y = a' = g^{x'} = g^{x+y} = g^x g^y \Rightarrow a = g^x$;
3) если на входе (p, g, a') алгоритм $\mathcal{A}$ выдаёт отказ, то алгоритм $\mathcal{B}$ останавливается с выдачей отказа.

Здесь $\mathcal{A}$ выдаёт отказ, если $(p, g, a') \in S_{\mathcal{A}}$. Величина $a' = ag^y$ при $y = 0, 1, \ldots, p-1$ пробегает все ненулевые элементы в $\mathrm{GF}(p)$, поэтому $\{(p, g, a') : y \in \{0, 1, \ldots, p-1\}\} = I_n$. Вероятность попадания (p, g, a') из I_n в $S_{\mathcal{A}}$ равна $\rho_n(S_{\mathcal{A}}) = \dfrac{|S_{\mathcal{A}} \cap I_n|}{|I_n|}$. Поскольку в генерическом алгоритме $\mathcal{A}$ множество $S_{\mathcal{A}}$ пренебрежимо, эта вероятность с ростом n приближается к 0 и при некотором n становится меньше $1/2$. Таким образом, при достаточно большом n алгоритм $\mathcal{B}$ останавливается и выдаёт отказ с вероятностью меньше $1/2$. ■

Следствие 4. Если для вычисления дискретного логарифма в поле $\mathrm{GF}(p)$ для произвольного p из экспоненциальной последовательности простых чисел не существует вероятностного алгоритма полиномиальной сложности, то для его вычисления не существует и генерического алгоритма полиномиальной сложности.

Литература

1. *Ахо А., Хопкрофт Дж., Ульман Дж.* Построение и анализ вычислительных алгоритмов. М.: Мир, 1979. 536 с.
2. *Хопкрофт Дж., Мотвани Р., Ульман Дж.* Введение в теорию автоматов, языков и вычислений. 2-е изд. М.: Изд. дом «Вильямс», 2002. 528 с.
3. *Myasnikov A. G. and Rybalov A. N.* Generic complexity of undeciable problems // J. Symbolic Logic. 2008. V. 73. No. 2. P. 656–673.
4. *Hamkins J. D. and Miasnikov A. G.* The halting problem is decidable on a set of asymptotic probability One // Notre Dame J. Formal Logic. 2006. V. 47. No. 4. P. 515–524.
5. *Rybalov A.* On the strongly generic undecidability of the halting problem // Theor. Computer Science. 2007. V. 377. No. 1–3. P. 268–270.
6. *Рыбалов А. Н.* О генерической сложности проблемы дискретного логарифма // Прикладная дискретная математика. 2016. № 3(33). С. 93–97.
7. *Быкова В. В.* Методы анализа и разработки параметризированных алгоритмов. Дисс. ... доктора физ.-мат. наук. Красноярск, 2012.

Содержание

Предисловие 3

1. Сложность алгоритмов 4

2. Асимптотические оценки сложности алгоритмов 6

3. Основные сложностные классы алгоритмов 8

4. Алгоритмические задачи 10

5. Алгоритмы разрешения 14

6. Неразрешимые задачи 15

7. Трудноразрешимые задачи 16

8. Машины Тьюринга 17

9. Классы задач $\mathcal{P}$ и $\mathcal{NP}$ 23

10. NP-полные задачи 23

11. NP-полнота задачи выполнимости 24

12. Другие NP-полные задачи 28

13. Генерическая сложность 30
    13.1. Генерическая разрешимость 30
    13.2. Генерическая разрешимость задачи останова ДМТ . 32
    13.3. Перечислимость 34
    13.4. Сильная генерическая неразрешимость задачи останова ДМТ 35
    13.5. Существование абсолютно неразрешимых задач . . 36
    13.6. Генерическая сложность дискретного логарифмирования 38

Литература 40

Учебное издание

АГИБАЛОВ Геннадий Петрович

ТЕОРИЯ ВЫЧИСЛИТЕЛЬНОЙ СЛОЖНОСТИ

Учебное пособие

Издание подготовлено в авторской редакции

Подписано к печати 19.11.2018. Формат $60 \times 84\frac{1}{16}$
Бумага для офисной техники. Гарнитура Times.
Усл. печ. л. 2,3.
Тираж 100 экз. Заказ № 3521.

Отпечатано на оборудовании
Издательского Дома
Томского государственного университета.
634050, г. Томск, пр. Ленина, 36
Сайт: http://publish.tsu.ru. E-mail: rio.tsu@mail.ru

ISBN 978-5-94621-768-2