Министерство образования и науки Российской Федерации
Государственное образовательное учреждение
высшего профессионального образования
«РОСТОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

М.Г. АДИГЕЕВ

ВВЕДЕНИЕ В ТЕОРИЮ СЛОЖНОСТИ

Методические указания для студентов
механико-математического факультета

Ростов–на–Дону
2004 г.

Печатается по решению учебно-методической комиссии
механико-математического факультета РГУ от

АННОТАЦИЯ

В данных методических указаниях изложены основы теории сложности алгоритмов и вычислений. Указания составлены на основе лекций, читаемых автором для студентов механико-математического факультета, специализирующихся по кафедре алгебры и дискретной математики.

Методические указания предназначены для студентов отделений «Прикладная математика» и «Защита информации» механико-математического факультета.

Автор: Адигеев М.Г.

ВВЕДЕНИЕ

Теория сложности вычислений — бурно развивающаяся область теоретической информатики (theoretical computer science) и охватывает как чисто теоретические вопросы, так и вопросы, непосредственно связанные с практикой. Среди наиболее важных приложений этой теории можно назвать способы построения и анализа эффективных алгоритмов, а также современные криптографические методы. Поэтому знакомство с основами теории сложности, безусловно, полезно любому, кто собирается серьезно заниматься практическим программированием или теоретическими исследованиями.

Данные методические указания содержат изложение базовых понятий и основных результатов теории сложности и соответствуют первой части курса лекций, читаемого автором для студентов механико–математического факультета. В будущем планируется выпуск методических указаний по следующим частям (сводимость задач, NP-полнота; вероятностные алгоритмы; приложения в криптографии).

1 СЛОЖНОСТЬ АЛГОРИТМОВ

Под алгоритмом обычно понимают четко определенную последовательность действий, приводящую через конечное число шагов к результату — решению задачи, для которой разработан алгоритм.

Основные свойства, присущие любому алгоритму:

1. массовость — алгоритм предназначен для решения задачи с некоторым множеством допустимых входных данных;
2. конечность — алгоритм должен завершаться за конечное число шагов (но это количество шагов может быть разным для разных входных данных).

Задачи могут быть сформулированы по-разному (дифференциальные уравнения, задачи на графах, задачи оптимизации и т.п.). Для того чтобы можно было строить единую теорию алгоритмов, необходимо свести разные формулировки задач к какому-то «единому знаменателю». Например, можно считать, что задача сводится к вычислению некоторой функции $F:X\to Y$. Ясно, что в таком виде можно сформулировать любую задачу. Но для некоторых задач функция F может быть выражена неявно. Например, для задачи поиска минимума функции φ на отрезке [0,1] имеем: X = множество функций, Y = [0,1], $F(\varphi) = x^*$: $\varphi(x^*) = \min\{\varphi(x): x \in [0,1]\}$.

Не для любой задачи можно построить алгоритм. Существуют алгоритмически неразрешимые задачи. Например: задача самоприменимости машины Тьюринга, задача об остановке алгоритма [8,9]. Еще один важный пример алгоритмически неразрешимой задачи — автоматическое доказательство теорем.

Замечание 1. Проблема алгоритмической разрешимости задачи тесно связана с требованием *массовости* алгоритма. Алгоритмическая неразрешимость задачи означает, что для любого алгоритма *A*, решающего данную задачу, существует набор входных данных *x*, на котором алгоритм работает неверно (либо возвращает неправильный результат, либо не останавливается).

Но даже если существует алгоритм, решающий задачу, это еще не значит, что мы сможем этим алгоритмом воспользоваться на практике для решения реальных задач. Потому что алгоритм может требовать для своей работы слишком много ресурсов. Например, если

решение задачи на самых современных компьютерах займет 10^{10} лет — очевидно, такой алгоритм для нас бесполезен.

Иными словами, недостаточно существования какого-нибудь алгоритма — должен существовать алгоритм, не требующий для своей работы слишком много ресурсов.

Определение 1. *Количественная характеристика потребляемых ресурсов, необходимых программе или алгоритму для работы (успешного решения задачи) — это и есть* сложность алгоритма.

Основные ресурсы: время (*временна́я сложность*) и объем памяти (*ёмкостная сложность*). Наиболее важной (критической) характеристикой является *время*.

Очевидно, что для разных экземпляров задачи (для разных входных данных) алгоритму может требоваться разное количество ресурсов.

С каждым экземпляром x задачи Z связывается определенное число (реже — набор чисел) $|x|$, называемое *длиной* или *размером входных данных (размером задачи)*. Размер задачи — это объем входных данных, необходимых для задания всех параметров задачи.

Однако для многих задач количество времени, необходимое для решения задачи, зависит не только от *размера* входных данных, но и от самих данных. То есть для решения задачи для двух входных данных x и y одинакового размера ($|x|=|y|$) алгоритм может тратить разное время. Поэтому для получения оценок временной сложности в зависимости от размера задачи определяют ее как *максимальное* время, затрачиваемое алгоритмом для входных данных длины n:

$T(n)=\max\{T(x): |x|=n\}$. Эта функция называется *сложностью* алгоритма *в худшем случае*.

Во многих случаях более важной для практики характеристикой алгоритма является его *сложность в среднем*. Формально сложность в среднем определяется так: $T_{ср}(n)=\sum T(x)p(x)$, где $p(x)$ — вероятность появления входных данных x, а суммирование ведется по всем возможным входным данным размера n. К сожалению, только для небольшого количества задач (например, для задачи сортировки) удается найти естественный способ определения вероятностей для входных данных. Поэтому при оценке сложности алгоритма обычно рассматривают его сложность в худшем случае.

Более того, обычно оценивают не точное значение функции сложности $T(n)$, а *порядок роста* этой функции [8], т.е. находят такую функцию $f(n)$, что $T(n) = O(n)$ при $n\to\infty$.

2 КЛАССЫ СЛОЖНОСТИ

В предыдущем параграфе мы ввели понятие «сложность алгоритма». Хотелось бы аналогичным образом определить и сложность **задачи** — например, как сложность самого эффективного (по времени или ёмкости) алгоритма, решающего эту задачу (для данных размера n). К сожалению, это невозможно. Доказано, что есть задачи, для которых *не существует* самого быстрого алгоритма, потому что любой алгоритм для такой задачи можно «ускорить», построив более быстрый алгоритм, решающий эту задачу. Это утверждение называют теоремой Блюма об ускорении [4]. Если отвлечься от технических деталей, то упрощенный вариант теоремы Блюма можно сформулировать следующим образом:

Теорема 1 (теорема Блюма об ускорении). *Существует такая алгоритмически разрешимая задача Z, что любой алгоритм A, решающий задачу Z, можно ускорить следующим образом: существует другой алгоритм A′, также решающий Z и такой, что* $T_{A'}(n) \le \log T_A(n)$ *для почти всех n.*

Замечание 2. Теорема Блюма не утверждает, что ускорение возможно для *любой* задачи. Более того, в дальнейшем мы увидим, что для задач, интересных с практической точки зрения, ускорение не возможно; для таких задач существует оптимальный (самый быстрый) алгоритм. Тем не менее, утверждение теоремы Блюма о существовании «неудобных» задач не позволяет определить универсальное (применимое ко всем задачам) понятие «оптимального алгоритма».

Поэтому в теории сложности использован другой подход — через классы сложности.

Определение 2. *Пусть* $f(n)$ *— некоторая функция, отображающая* $\boldsymbol{N}$ *в* $\boldsymbol{N}$. Класс сложности $C(f(n))$ *— это множество всех задач, для которых существует хотя бы один алгоритм, сложность которого не превышает* $O(f(n))$.

Это определение в некотором смысле условно — обозначение $C(f(n))$ никогда не применяют. Почему? Потому что для задания реального класса задач необходимо еще уточнить:

- что мы понимаем под «алгоритмом»;
- какая сложность (временная, емкостная или какая-нибудь еще) нас интересует.

При разных ответах на эти вопросы получатся разные классы задач, и для каждого класса используется специальное обозначение. В теории сложности под «алгоритмом» обычно понимают ту или иную разновидность машины Тьюринга. Перейдем к рассмотрению различных типов алгоритмов (машин Тьюринга) и соответствующих им классов сложности.

3 ДЕТЕРМИНИРОВАННЫЕ АЛГОРИТМЫ

В данном параграфе мы рассмотрим простейший вариант машины Тьюринга — детерминированную МТ. Прилагательное «*детерминированная*» пока будем опускать — его значение будет объяснено позже, при рассмотрении *недетерминированных* машин.

3.1 Детерминированная однолентоная машина Тьюринга

ОПРЕДЕЛЕНИЕ 3. Алфавитом *называется произвольное непустое счетное множество. Обычно рассматривают конечные алфавиты. Элементы алфавита называются* символами *или* буквами. Словом *в алфавите A называется конечная последовательность букв из этого алфавита. Количество букв в слове x называется* длиной *слова и обозначается* $|x|$. A^* = *множество всех слов над алфавитом A.* A^k = *множество всех слов длины k.*

ОПРЕДЕЛЕНИЕ 4. Машина Тьюринга *(MT) — это четверка* $M = (Q, A, S, \Pi)$*, где A — «ленточный» алфавит (содержит специально*

выделенный символ $\wedge$ — *«пробел»),* $Q = \{q_0, q_1, \ldots, q_m\}$ — *алфавит состояний,* $S = \{-1, 0, +1\}$ — *алфавит сдвигов, и* Π — *программа, представляющая собой отображение* $Q \times A \to Q \times A \times S$.

Принцип работы МТ подробно описан в [2, §1.6] и [9].

Формализовав таким образом интуитивное понятие «алгоритм», мы можем четко определить временную и емкостную сложность: $T_M(x)$ — количество шагов, сделанных машиной M при обработке входа x, $S_M(x)$ — количество ячеек на ленте, на которых побывала головка машины M при обработке входа x. Если на входе x машина зацикливается, то значения $T_M(x)$ и $S_M(x)$ не определены. После этого указанным выше образом определяются функции $T_M(n)$ и $S_M(n)$.

Рассмотренное определение задает простейшую модель алгоритма — *детерминированную одноленточную* машину Тьюринга.

3.2 Многоленточные машины

Наше определение временной и ёмкостной сложности алгоритма опирается на простейшую модель МТ. В связи с этим возникают следующие вопросы. Насколько сильно временная и ёмкостная сложности зависят от вычислительной модели? Изменится ли сложность алгоритма, если мы расширим модель — скажем, «разрешим» машине Тьюринга иметь более одной ленты?

Введем новое определение.

Определение 5. *Пусть* k — *целое число,* $k \geq 1$. k–ленточная машина Тьюринга — *это пятерка* $M = (k, A, Q, S, \Pi)$, *где* $\Pi: Q \times A^k \to Q \times (A \times S)^k$.

Принцип работы многоленточных машин в целом такой же, как и у одноленточных. Отличия связаны с количеством лент. Очередной шаг многоленточной машины определяется символами, расположенными в текущих ячейках на *всех* лентах, т.е. набором (q, $a_1,\ldots, a_k)\in Q\times A^k$. По этому набору определяются выполняемые действия: (q', b_1, $s_1,\ldots, b_k$, $s_k)\in Q\times(A\times S)^k$. Эти действия также производятся на всех лентах: на i-й ленте в текущую ячейку записывается символ b_i и головка смещается в соответствии с s_i. Обратите внимание на то, что головки на лентах перемещаются независимо друг от друга.

Временная сложность многоленточной машины определяется точно так же, как и для одноленточной. В определении емкостной сложности есть небольшое изменение: емкостная сложность k-ленточной МТ M на входе x $S_M(x)$ определяется как *максимум* количества ячеек, на которых при обработке входа x побывали головки машины M на всех лентах. Очевидно, что это определение согласуется с данным ранее определением для k=1.

В будущем нам потребуется следующее вспомогательное утверждение (его доказательство остается в качестве упражнения).

ТЕОРЕМА 2. *Для любой машины Тьюринга выполняется неравенство $S(n)\le T(n)$, т.е. емкостная сложность не превышает временную*.

3.3 Эквивалентность машин

Очевидно, что понятие *k*–ленточной МТ шире, чем понятие «обычной» одноленточной машины. Но насколько увеличилась «вычислительная мощь» машин Тьюринга за счет добавления лент? Более точно, нас интересуют следующие вопросы.

1. *Качественные* отличия: есть ли такие задачи, которые можно решать с помощью *k*–ленточных машин (при $k>1$), но нельзя решить с помощью одноленточной машины?
2. *Количественные* отличия. Очевидно, что за счет использования дополнительных лент можно получить выигрыш в скорости. Как сильно можно ускорить решение задачи, добавляя ленты?

Для того чтобы ответить на эти вопросы, введем следующие определения [8].

Определение 6. (*k*+1)*–ленточная МТ M′ с программой w* симулирует *k–ленточную машину M, если для любого набора входных слов* (x_1, x_2, …, x_k) *результат работы M′ совпадает с результатом работы M на этих же входных данных. Предполагается, что вначале слово w записано на* (*k*+1)*-й ленте M′. Под результатом понимается состояние первых k лент МТ в момент остановки, а если на данном входе M не останавливается, то симулирующая ее машина также не должна останавливаться на данном входе*.

Определение 7. (*k*+1)*–ленточная МТ M* называется* универсальной машиной Тьюринга для *k*–ленточных машин, *если для любой k-ленточной машины M существует программа w, на которой M* симулирует M*.

Обратите внимание: в определении универсальной МТ одна и та же машина M' должна симулировать разные k-ленточные машины (на разных программах w).

Рассмотрим следующую теорему [8].

ТЕОРЕМА 3. *Для любого $k \geq 1$ существует универсальная* (k+1)*–ленточная машина Тьюринга*.

Доказательство. Теорему докажем конструктивно, т.е. покажем, как можно построить требуемую универсальную машину **M***. Рассмотрим лишь общую схему построения, опустив сложные детали. Основная идея заключается в том, чтобы на дополнительную (k+1)-ю ленту разместить описание симулируемой машины Тьюринга и использовать это описание в процессе симулирования.

Пусть $M = (k, Q, A, S, П)$ — произвольная k–ленточная машина Тьюринга. Не нарушая общности можно считать, что алфавит A содержит символы ‘0’, ‘1’ и вспомогательный символ ‘*’ (и, возможно, какие-то другие символы). Каждое состояние $q \in Q$ можно закодировать двоичным словом фиксированной длины r ($r \geq \log_2|Q|$). Очевидно также, что сдвиги $s \in S$ можно закодировать с помощью двух бит (т.е. двоичных слов длины 2). Рассмотрим одну ячейку из табличного представления программы машины M. Пусть эта ячейка соответствует состоянию q и обозреваемым символам a_i (на i-й ленте текущим символом является a_i, i=1,…,k); и пусть машина должна в этом случае перейти в состояние q', напечатать символы b_i и выполнить смещения s_i (i=1,…,k). Такую ячейку можно закодировать словом

$$qa_1\, a_2 \ldots a_k q' b_1\, b_2 \ldots b_k s_1\, s_2 \ldots s_k\,.$$

Это слово состоит из $2r+4k$ символов алфавита A. В дальнейшем каждое такое слово будем называть *блоком*. Закодируем таким образом все ячейки в программе машины M и запишем их в виде последовательности блоков на $(k+1)$-ю ленту машины.

Для того чтобы просимулировать поведение машины M, нам необходимо не только иметь программу этой машины, но и отслеживать ее текущее состояние. Текущее состояние машины M также будем записывать на $(k+1)$-ю ленту. Для того чтобы различать запись программы и запись текущего состояния, будем разделять их символом '*'. Например, слева от символа '*' расположим слово α — закодированное представление текущего состояния машины M, а справа — слово π, представляющее собой закодированное представление программы этой машины.

Процедура моделирования машины M включает в себя следующие шаги (выполняемые машиной **M***):

1. Найти в слове π блок, соответствующий «текущему состоянию» α и текущим символам на первых k лентах.
2. На первых k лентах выполнить действия, задаваемые найденным блоком, т.е. изменить символы в текущих ячейках и сместить головки.
3. Изменить «текущее состояние» симулируемой машины, записав на $(k+1)$-й ленте слева от символа '*' вместо слова α слово α', считанное из найденного блока.
4. Проверить, является ли «состояние» α' заключительным. Если является, то завершить работу, иначе перейти к п.1.

Очевидно, что каждый из шагов 1–4 можно «запрограммировать» в виде машины Тьюринга. Искомая

универсальная машина **M*** представляет собой их композицию (см. [9]) ■

Доказанная теорема позволяет нам подойти к ответу на поставленные ранее два вопроса — убедиться, что количество лент у МТ не имеет принципиального значения ни с точки зрения вычислительной мощности, ни с точки зрения временной сложности. Напомню, что «принципиальное значение» для нас имеет только различие между полиномиальной и неполиномиальной (экспоненциальной) сложностью.

Часто нас интересует только «выходное» значение, расположенное на последней ленте. В этом случае имеет смысл ослабить понятие симулирования.

Определение 8. *Будем говорить, что машина Тьюринга M* вычисляет частичную функцию $f:A^*\to A^*$, *если для любого* $x\in A^*$, *записанного на первую ленту машины M:*

- *если* $f(x)$ *определено, то M останавливается, и в момент остановки на последней ленте машины записано слово* $f(x)$;
- *если* $f(x)$ *не определено, то машина M не останавливается*.

Определение 9. *Будем говорить, что машины M и M´* эквивалентны, *если они вычисляют одну и ту же функцию*.

Понятие эквивалентности «слабее», чем симулирование: если машина M' симулирует машину M, то машина M' эквивалентна M; обратное, вообще говоря, неверно. С другой стороны, для симулирования требуется, чтобы у M' было как минимум столько же лент, сколько и у M, в то время как для эквивалентности это не

обязательно. Именно это свойство позволяет нам сформулировать и доказать следующую теорему [8].

ТЕОРЕМА 4. *Для любой k-ленточной машины M, имеющей временную сложность T(n), существует эквивалентная ей одноленточная машина M′ с временной сложностью* $T'(n) = O(T^2(n))$.

Доказательство. Сначала рассмотрим идею, лежащую в основе алгоритма моделирования. В отличие от задачи симулирования k-ленточной машины с помощью $(k+1)$-ленточной (Теорема 3), у машины M' имеется только одна лента, с помощью которой необходимо отслеживать состояние k лент моделируемой машины M. Для этого «упакуем» все k лент машины M в одну ленту машины M'. Под «упаковкой» будем понимать взаимно однозначное отображение ячеек всех лент машины M в ячейки ленты машины M' по следующему правилу: i-й ячейке j-й ленты M соответствует ячейка с номером $2(ki+j-1)$ на ленте M'. Таким образом, для хранения входных и промежуточных данных будут использоваться ячейки с четными номерами. В ячейках с нечетными номерами будем запоминать положение головок на лентах моделируемой машины: если у машины M головка на ленте j находится в ячейке i, то на ленте машины M' в ячейке $2(ki+j-1)$ стоит символ ‘1’, иначе — пробел.

При таком способе упаковки возникает одна чисто техническая проблема: на ленте машины M' внутри записанного слова окажутся пробелы, а это противоречит определению машины Тьюринга. Для того чтобы преодолеть эту проблему, введем дополнительный символ ‘*’ (отсутствующий в алфавите машины M) и будем использовать его вместо пробела внутри слова (в том числе и в ячейках с нечетными

номерами — в качестве признака того, что соответствующая ячейка машины M не является текущей).

Таким образом, работа машины M' состоит из трех фаз:

1. Начальное преобразование входного слова x.
2. Моделирование работы машины M.
3. Завершающее преобразование результата.

Первую фазу мы уже рассмотрели. Во время второй фазы машина M' моделирует каждый шаг машины M. При этом на каждом шаге M' за счет своих состояний помнит текущее состояние машины M и номер обрабатываемой ленты. За один проход по своей ленте машина M' выясняет, какие символы считываются каждой головкой машины M и определяет, что необходимо сделать — в какое состояние нужно перейти, что записать на каждую из лент и как сместить головки на лентах. Все это M' «запоминает» с помощью своего состояния. Затем M' еще раз проходит по своей ленте и выполняет необходимые преобразования.

Третья фаза алгоритма представляет собой преобразование «сжатия», обратное первой фазе, с той разницей, что теперь мы оставляем только символы, записанные на последней (k-й) ленте машины M.

Очевидно, что работающая по такому алгоритму машина M' действительно эквивалентна машине M. Для завершения доказательства нам осталось оценить временную сложность M'. Фазы 1 и 3 требуют $O(S(n))$ шагов, где $S(n)$ — емкостная сложность машины M. Во время фазы 2 машина M' должна промоделировать $T(n)$ шагов машины M, сложность моделирования каждого шага равна $O(S(n))$, потому что на каждом шаге M' дважды сканирует все слово,

записанное на ее ленте. Получаем, что временная сложность машины M' равна $O(T(n)S(n))$, что с учетом теоремы 2 равно $O(T^2(n))$ ■

Определение 10. $DTIME(f(n))$ — *это класс задач, для каждой из которых существует детерминированная МТ, решающая эту задачу с временной сложностью* $O(f(n))$.

Напомню, что в строгой формулировке под решением задачи мы понимаем вычисление некоторой функции.

Доказанные теоремы (об универсальной МТ и о моделировании произвольной машины с помощью одноленточной) обеспечивают универсальность данного определения (каждая алгоритмически разрешимая задача принадлежит некоторому классу $DTIME(f(n))$).

Определение 11. Определим также классы:

$P = PTIME = \cup_{k>0} DTIME(n^k)$ — *класс задач, разрешимых за* полиномиальное *время*.

$EXPTIME = \cup_{k>0} DTIME(2^{(n^k)})$ — *класс задач, решаемых за* экспоненциальное *время*. *Здесь выражение* 2^{x} *означает «2 в степени x».*

В силу теоремы 4 определение классов PTIME и EXPTIME не зависит от количества лент в машине.

Класс P — это класс эффективно решаемых задач. Алгоритм, имеющий полиномиальную временную сложность, называется *эффективным*.

Задача, для которой в настоящее время не известен эффективный алгоритм или для которой доказано отсутствие такого алгоритма, называется *трудноpешаемой*.

3.4 Иерархия классов DTIME

Очевидно, что если $g(n)>f(n)$ для любого n, то любую задачу, которая может быть решена за время $O(f(n))$, можно решить и за время $O(g(n))$. То есть $DTIME(f(n))\subseteq DTIME(g(n))$. Но может быть, есть такие задачи, которые можно решить за время $O(g(n))$, но нельзя решить за время $O(f(n))$? «Здравый смысл» подсказывает, что в «общем случае» именно так и должно быть. Но в математике не всегда можно полагаться на «здравый смысл». При более тщательном анализе видно, что и в данном вопросе не все так просто. Например, если функция g превышает f лишь в константное число раз (например, $g(n)=c\cdot f(n)$, $c>1$), то классы $DTIME(g(n))$ и $DTIME(f(n))$ все равно совпадают, потому что в их определении используется выражение $O(\cdot)$. Интересно, что эти классы были бы равны даже в том случае, если бы мы в определении класса DTIME не использовали обозначение $O(\cdot)$, т.е. определили бы $DTIME(f(n))$ как класс задач, решаемых за время $f(n)$). Это следует из так называемой «теоремы о линейном ускорении»:

ТЕОРЕМА 5. (Теорема о линейном ускорении). *Для любой машины M с временной сложностью $T(n)$ и любой константы $c>0$, существует эквивалентная машина Тьюринга M' с временной сложностью $T'(n) = cT(n)+n$.*

Доказательство этой теоремы можно найти в [6].

Поэтому для того, чтобы класс DTIME($g(n)$) оказался «шире» класса DTIME($f(n)$), функция g должна расти *значительно быстрее* функции f. Точную «расшифровку» выражения «значительно быстрее» содержит так называемая «теорема об иерархии». Прежде чем перейти к формулировке и доказательству этой теоремы, введем несколько определений и обозначений.

Введем обозначение: для двух числовых функций f и g пишут $g(n) = \omega(f(n))$, если $lim(f(n)/g(n)) = 0$ при $n \to \infty$.

Определение 12. *Функция f называется* конструируемой по времени, *если существует машина Тьюринга M, которая для данного входного слова длины n останавливается ровно через* $f(n)$ *шагов*.

Определение 13. *Функция f называется* конструируемой по ёмкости, *если существует машина Тьюринга M, которая для данного входного слова длины n просматривает ровно* $f(n)$ *ячеек на одной из своих лент и не более* $f(n)$ *ячеек на остальных лентах*.

Все «обычные» функции, встречающиеся на практике (например: n, n^k, 2^n) являются конструируемыми как по времени, так и по ёмкости. Любая конструируемая по времени функция является также конструируемой по ёмкости (докажите это самостоятельно).

Определение 14. *Рассмотрим машину Тьюринга M, имеющую два заключительных состояния* q^Y *(*допускающее *состояние) и* q^N *(*отвергающее *состояние) и произвольное слово* $x \in A^*$.

- *Если M на входе x через конечное число шагов останавливается в состоянии q^Y, то будем говорить, что машина M* допускает *строку x.*
- *Если M на входе x останавливается в состоянии q^N или «зацикливается», то будем говорить, что машина M* отвергает *строку x.*

Определение 15. *Через L(M) обозначим множество всех слов $x \in A^*$, допускаемых машиной Тьюринга M.*

Определение 16. *Пусть L — язык над алфавитом A. Говорят, что машина M* распознает *язык L, если L(M)= L.*

Определение 17. *Пусть f — вычислимая конструируемая по емкости функция. Машина $\boldsymbol{M}_f(\cdot,\cdot)$, принимающая 2 входных строки, называется* симулятором с ограничением по времени, *если для любой поданной ей на вход машины M и любого входного слова x, выполняются условия:*

- *если M допускает строку x за f(n) шагов, то $\boldsymbol{M}_f$ также допускает строку x;*
- *иначе $\boldsymbol{M}_f$ отвергает строку x.*

ТЕОРЕМА 6. *Пусть f и g — две вычислимые конструируемые по времени функции, и $g(n) = \omega(f(n) \log f(n))$. Тогда существует симулятор с ограничением по времени $\boldsymbol{M}_f$, работающий за время* $O(g(n))$.

Доказательство. Мы не будем рассматривать строгое доказательство этой теоремы, чтобы не углубляться в технические детали. Рассмотрим лишь общую идею построения подобного симулятора, как это сделано в [6, §3.2].

Машина $\boldsymbol{M}_f$ содержит 4 ленты. Первая лента предназначена для подсчета количества шагов. На этой ленте еще до начала собственно «симулирования» машина $\boldsymbol{M}_f$ помечает (с помощью любого символа, отличного от «пробела») $f(|x|)$ ячеек. Эта операция допустима, поскольку функция f по условию теоремы является конструируемой по времени — а значит, и по ёмкости.

На второй ленте хранится содержимое рабочих лент симулируемой машины, а также запоминается положение ее головок (например, так, как это сделано в доказательстве теоремы 4). На третьей ленте записывается строка, представляющая программу симулируемой машины и ее текущее состояние (см. доказательство теоремы 3). Наконец, четвертая лента предназначена для хранения рабочей информации самой машины $\boldsymbol{M}_f$.

Как уже говорилось выше, машина $\boldsymbol{M}_f$ первым делом «заводит таймер», отмечая ровно $f(n)$ ячеек на одной из своих лент. Далее $\boldsymbol{M}_f$ последовательно симулирует поведение входной машины так же, как описано в доказательстве теоремы 3. Но, дополнительно, после моделирования очередного шага входной машины, $\boldsymbol{M}_f$ стирает один символ на своей первой ленте и проверяет, остались ли еще символы (отличные от пробела) на этой ленте. Если нет («время вышло»), то $\boldsymbol{M}_f$ останавливается и отвергает входное слово x. Если же моделируемая машина остановится до того, как закончатся символы на «таймерной»

ленте, то $\boldsymbol{M}_f$ также останавливается и возвращает тот же результат (допускает или отвергает слово x).

Описанная процедура симулирования дает оценку $O(f^3(n))$ для временной сложности машины $\boldsymbol{M}_f$. Для того чтобы получить указанную в теореме оценку, надо немного «оптимизировать» эту процедуру (но мы, как и договаривались, не будем этого делать). ■

Теперь мы, наконец, готовы сформулировать и доказать основную теорему данного параграфа (см. также, [2, стр.471] и [6, §3.3]).

ТЕОРЕМА 7. (Теорема об иерархии). *Пусть f и g — две вычислимые конструируемые по времени функции, и $g(n) = \omega(f(n) \log f(n))$. Тогда класс $DTIME(f(n))$ строго вложен в класс $DTIME(g(n))$.*

Доказательство. Для доказательства будет использована процедура диагонализации (в курсе математического анализа аналогичным методом доказывалась несчетность множества вещественных чисел). Мы «построим» язык, принадлежащий классу $DTIME(g(n))$, но не принадлежащий классу $DTIME(f(n))$. К сожалению, такое «построение» будет неконструктивно, т.е. мы не сможем указать реальную (реализуемую неким алгоритмом) процедуру построения такого языка.

Через $C(x)$ обозначим машину $\boldsymbol{M}_f(x,x)$, то есть на вход симулятора подаются два одинаковых слова. Через $D(x)$ обозначим инверсию машины $C(x)$, то есть машину, работающую так же, но выдающую противоположный ответ (если C допускает слово x, то D отвергает x, и наоборот). Здесь важным является тот факт, что

существует ограничение сверху на время работы $\boldsymbol{M}_f(x,x)$ (см. теорему 6), а значит и на время работы C — иначе машины D не существовало бы, поскольку ей пришлось бы решать задачу об остановке машины C, а эта задача в общем виде алгоритмически неразрешима.

Через L обозначим язык, распознаваемый машиной D («диагонализирующая» машина), т.е. $L=L(D)$. Покажем, что это и есть требуемый язык.

Согласно теореме 6, D работает время $O(g(n))$, а значит, язык L принадлежит классу $\mathrm{DTIME}(g(n))$.

С другой стороны, для любой машины M, распознающей язык из класса $\mathrm{DTIME}(f(n))$, по построению машины D справедливо неравенство $M(M) \neq D(M)$. А это означает, что язык L отличается от любого языка из класса $\mathrm{DTIME}(f(n))$: от языка, распознаваемого машиной M язык L отличается в строке, представляющей собой закодированное представление машины M. Поэтому, L не принадлежит классу $\mathrm{DTIME}(f(n))$. ■

Сформулируем также два важных следствия из теоремы об иерархии (попробуйте доказать эти утверждения самостоятельно).

СЛЕДСТВИЕ 1. *Для любого целого числа k (k>0) выполняется*

$$\mathrm{DTIME}(n^k) \subset \mathrm{DTIME}(n^{k+1}).$$

СЛЕДСТВИЕ 2. $\mathrm{PTIME} \subset \mathrm{EXPTIME}$.

Здесь знак $\subset$, в отличие от знака $\subseteq$, означает *строгое* вложение.

4 НЕДЕТЕРМИНИРОВАННЫЕ АЛГОРИТМЫ

4.1 Недетерминированные машины Тьюринга

До сих пор мы рассматривали так называемые «детерминированные» алгоритмы. Их «детерминированность» заключается в том, что на каждом шаге имеется лишь одно допустимое действие, зависящее от входных данных и от текущего «внутреннего» состояния. В терминах машин Тьюринга детерминированность отражается в том, что программа П является *отображением*, то есть каждой паре (a,q) ставит в соответствие однозначно определенную тройку (a',q',s).

Теперь мы познакомимся с *недетерминированными* алгоритмами, играющими важную роль в теории сложности вычислений. Недетерминированные алгоритмы являются, в некотором смысле, обобщением детерминированных: на некоторых (возможно — на всех) шагах у недетерминированного алгоритма имеется выбор действия из несколько вариантов. Причем этот выбор не зависит ни от каких внутренних или внешних факторов (ничем *не детерминирован*). Для того чтобы формализовать понятие недетерминированного алгоритма, рассмотрим недетерминированные машины Тьюринга [2, 6].

Определение 18. k-ленточной недетерминированной машиной Тьюринга *называется пятерка* $M = (A, Q, S, П, q_0)$. *Значения всех компонент пятерки — такие же, как и в случае k-ленточной детерминированной МТ, за исключением того, что программа П представляет собой не отображение, а отношение, заданное на множестве* $(Q\times A^k)\times(Q\times(A\times S)^k)$.

Если вы забыли, в чем заключается разница между отображением и отношением, рекомендую освежить свои знания по курсу дискретной математики, например [9].

Принцип работы НМТ в целом такой же, как и у ДМТ. Но, как уже говорилось, для некоторых конфигураций машины (т.е. наборов $(q,\ a_1,...,a_k)$) может существовать несколько конфигураций $(q',\ a'_1,...,a'_k,\ s'_1,...,s'_k)$, связанных отношением П с текущей конфигураций, т.е. таких, что $(q,\ a_1,...,a_k,\ q',\ a'_1,...,a'_k,\ s'_1,...,s'_k)\in$ П. В этом случае машина в качестве следующей конфигурации выбирает *любой* такой набор. Однако НМТ не может самостоятельно комбинировать элементы разных наборов, например, выбрать следующее текущее состояние q' из одного набора, а новые ленточные символы $a'_1,...,a'_k$ — из другого.

Для недетерминированных алгоритмов обычно рассматривают только одну форму задач — задачи распознавания языков. Рассмотрим подробнее определение принимаемых и отвергаемых слов для НМТ. Очевидно, что при обработке любого входного слова x недетерминированная машина M может пройти разные последовательности конфигураций (за счет того, что на некоторых шагах выбор следующей конфигурации недетерминирован). Считают, что НМТ M *допускает* входное слово x, если *хотя бы одна* такая последовательность конфигураций приводит к допускающему состоянию q^Y (такая последовательность называется *допускающей*). В противном случае, т.е. если *ни одна* последовательность конфигураций не приводит к допускающему состоянию, машина M *отвергает* слово x. Таким образом, в отличие от детерминированных машин, ситуации допуска или отвержения входного слова не симметричны.

Определение языка, распознаваемого НМТ, полностью аналогично соответствующему определению для ДМТ.

Определение 19. *Говорят, что НМТ M имеет* временную сложность *T*(*n*), *если для всякого допускаемого входного слова длины n найдется последовательность, состоящая не более чем из T*(*n*) *шагов, приводящая в допускающее состояние.*

Определение 20. *Говорят, что НМТ M имеет* ёмкостную сложность *S*(*n*), *если для всякого допускаемого входного слова длины n найдется последовательность, приводящая в допускающее состояние,в которой число просмотренных ячеек на каждой ленте не превышает S*(*n*).

Для недетерминированных машин Тьюринга справедлив аналог теоремы 4:

ТЕОРЕМА 8. *Для любой k-ленточной недетерминированной машины M, имеющей временную сложность T*(*n*), *существует однолenточная НМТ M´, моделирующая M с временной сложностью* $T'(n) = O(T^2(n))$.

Доказательство этой теоремы полностью аналогично доказательству теоремы 4.

4.2 Классы NTIME, NP и co-NP

Определение 21. Определим классы задач по сложности относительно недетерминированных алгоритмов:

NDTIME($f(n)$) — *это класс задач, для каждой из которых существует недетерминированная однолентоная МТ, решающая эту задачу с временной сложностью* O($f(n)$)).

NP = NPTIME = $\cup_{k>0}$NDTIME(n^k) — *класс задач, решаемых недетерминированными алгоритмами за* полиномиальное *время*.

NEXPTIME = $\cup_{k>0}$NDTIME(2^(n^k)) — *класс задач, решаемых недетерминированными алгоритмами за* экспоненциальное *время*.

Очевидно, что детерминированные машины Тьюринга являются частным случаем недетерминированных. В частности, P $\subseteq$ NP.

Как и в случае перехода от одноленточных ДМТ к многоленточным, перед нами встают вопросы о том, насколько «мощнее» НМТ по сравнению с ДМТ. Более конкретно:

1. Есть ли такие задачи, которые можно решать с помощью НМТ, но нельзя решить с помощью ДМТ?
2. Поскольку НМТ может «одновременно» выполнять все варианты вычислений, недетерминированная машина, вообще говоря, может решить некоторую задачу быстрее, чем детерминированная машина. Но насколько быстрее? Теорема о линейном ускорении (теорема 5 на стр. 18) показывает, что линейное ускорение возможно и без привлечение недетерминированности. Для практических целей важны эффективные (полиномиальные) алгоритмы, поэтому прежде всего интересен ответ на такой вопрос: могут ли недетерминированные алгоритмы эффективно (т.е. за

полиномиальное время) решать задачи, неразрешимые за полиномиальное время с помощью ДМТ? С помощью введенных обозначений этот вопрос можно переформулировать так: есть ли задачи, принадлежащие классу NP, но не принадлежащие классу P, или же выполняется равенство P=NP?

Рассмотрим эти вопросы по порядку.

ТЕОРЕМА 9. *Для любой НМТ M^N, распознающей язык L за время $T(n)$ найдется ДМТ M^D и константа c, такие, что M^D также распознает язык L, но за время $O(c^{T(n)})$.*

Доказательство. Для доказательства теоремы покажем, как можно построить требуемую ДМТ M^D. Машина M^D будет моделировать работу машины M^N, перебирая все возможные варианты вычислений.

Для простоты рассмотрим вариант одноленточной НМТ. Так как для каждой пары (q,a) имеется конечное количество вариантов следующего шага, и таких пар тоже конечное количество, то существует константа d, ограничивающая число вариантов следующего шага для всех пар (q,a). Перенумеруем такие варианты для каждой пары, номер варианта будет находиться в интервале от 1 до d.

По условию, при обработке входного слова длины n машина M^N делает не более $T(n)$ шагов. Поскольку начальное состояние фиксировано, и после каждого шага имеется не более d вариантов следующего шага, любую такую последовательность можно закодировать словом в алфавите $\{1, ..., d\}$ длины не более $T(n)$.

Количество таких слов равно $1+d+d^{2}+...+d^{T(n)} < (d+1)^{T(n)}$. Машина M^D последовательно генерирует все такие слова и для каждого слова w моделирует работу машины M^N при условии, что на i-м шаге (i=1,...T(n)) выбирается w_i-й вариант следующего шага. Если для какого-то слова w моделирование показывает, что машина M^N допускает входное слово x, M^D прекращает перебор и также допускает это слово. Если же после перебора всех слов длины не более T(n) входное слово не было допущено, M^D отвергает его.

Для каждого слова w моделирование работы M^N требует O(T(n)) шагов. Порождение очередного слова w также занимает O(T(n)) шагов. Поэтому весь процесс моделирования работы НМТ M^N на ДМТ M^D потребует время $O(T(n)\ (d+1)^{T(n)})$. Эта величина не превосходит $O(c^{T(n)})$ для некоторого c. ■

Таким образом, на 1-й вопрос (стр. 27) мы можем ответить отрицательно: любая задача, которую может решить НМТ, разрешима и с помощью ДМТ.

Ответ на 2-й вопрос, к сожалению, пока неизвестен. Большинство исследователей полагают, что классы P и NP различны (P строго вложен в NP), но ни доказать, ни опровергнуть это предположение пока никому не удалось.

Важным отличием классов сложности для ДМТ и НМТ является то, что классы для недетерминированных машин незамкнуты относительно операции дополнения языка. Рассмотрим это более подробно. Для языка L его *дополнение* $\overline{L}$ — это множество всех слов (в том же алфавите), не входящих в язык L.

Определение 22. *Для любого класса C класс co–C* = $\{L : \overline{L} \in C\}$ *будем называть* сопряженным *классом для C*.

Рассмотрим произвольный класс DTIME($f(n)$) и любой язык L из этого класса (см. также [8, стр. 44]). По определению, для L должна существовать ДМТ M, распознающая L за полиномиальное время. Инвертируем M, то есть получим машину M′, которая работает так же, как и M, но переходит в отвергающее состояние тогда, когда M переходит допускающее, и наоборот. Очевидно, что M′ допускает $\overline{L}$, причем делает это за полиномиальное время. Таким образом, мы доказали, что для любого класса DTIME($f(n)$) сопряженный к нему класс co-DTIME($f(n)$) совпадает с самим DTIME($f(n)$). И как следствие, P = co–P.

Для классов NTIME($f(n)$) подобные рассуждения не проходят, потому что ситуации, в которых НМТ допускает или отвергает входное слово, не симметричны: слово допускается, если существует *хотя бы одна* допускающая последовательность шагов, и отвергается, если *все* последовательности шагов приводят в отвергающее состояние. Вопрос о том, совпадают ли классы NTIME($f(n)$) и co–NTIME($f(n)$) (а также классы NP и co–NP), в настоящее время открыт. Однако нетрудно убедиться в том, что $P \subseteq NP \cap co–NP$ (докажите это самостоятельно).

4.3 Альтернативное определение класса NP

Класс NP можно определить не только через недетерминированные машины Тьюринга (определение 21). Введем альтернативное определение, которое удобно использовать для изучения свойств задач из NP и при доказательстве теорем.

Определение 23. *Язык L принадлежит классу* NP *тогда и только тогда, когда существует детерминированная машина Тьюринга M и полином p, такие что:*

- *для любого слова* $x \in L$ *существует слово–*сертификат *y,* $|y| \le p(|x|)$*, при котором машина M допускает пару (x,y) за полиномиальное время;*
- *ни для какого* $x \notin L$ *подобного сертификата не существует*.

Покажем, что определения 21 и 23 эквивалентны, то есть определяют один и тот же класс языков.

ТЕОРЕМА 10. *Определения* 21 *и* 23 *эквивалентны*.

Доказательство. (21) $\Rightarrow$ (23). Выберем произвольный язык L, принадлежащий классу NP в соответствии с определением 21. Это значит, что существует НМТ M^N, распознающая язык L за полиномиальное время. В качестве детерминированной машины, упоминаемой в определении 23, выберем M^D из доказательства теоремы 9. Тогда для любого слова *x* сертификатом будет закодированное представление допускающей последовательности шагов (слово *w* из доказательства теоремы 9). Длина сертификата ограничена полиномом от $|x|$ в силу того, что время работы M^N полиномиально. Очевидно, что такой сертификат существует только для слов из L. Таким образом, для L выполняются все условия определения 23.

(23) $\Rightarrow$ (21). Предположим, что для языка L выполняются условия определения 23. Построим недетерминированную МТ M^N следующим образом. M^N имеет две ленты. На первой ленте записано

входное слово x. На второй ленте M^N недетерминированно записывает слово z (длина z не превышает $p(|x|)$) и затем моделирует работу машины M на входе (x,z). Если $x \in L$, то M^N допустит x, потому что в этом случае найдется допускающая последовательность шагов — та самая, при которой слово z совпадет с сертификатом для x. Если же $x \notin L$, то при всех z M отвергает пару (x,z) — следовательно, M^N также отвергнет x. Таким образом, L удовлетворяет определению 21. ■

5 КЛАССЫ ПО ЕМКОСТНОЙ СЛОЖНОСТИ

По аналогии с классами DTIME, P и NP, можно ввести аналогичные классы по ёмкостной сложности. Сразу будем рассматривать именно задачи распознавания языков.

Определение 24. $DSPACE(f(n))$ — *это класс языков, для каждого из которых существует детерминированная МТ, распознающая этот язык с ёмкостной сложностью* $O(f(n))$.

$PSPACE = \cup_{k>0} DSPACE(n^k)$ — *класс языков, распознаваемых с полиномиальной ёмкостью*.

Определение 25. $NSPACE(f(n))$ — *это класс языков, для каждого из которых существует недетерминированная МТ, распознающая этот язык с ёмкостной сложностью* $O(f(n))$.

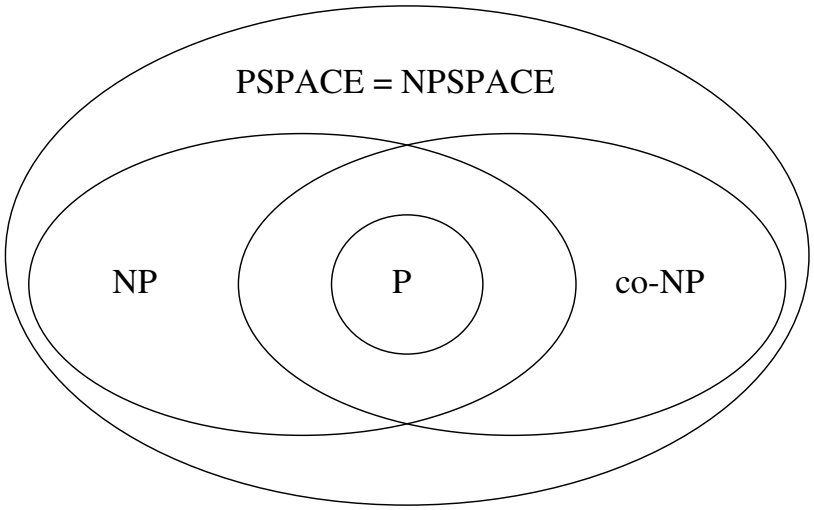
$NPSPACE = \cup_{k>0} NSPACE(n^k)$ — *класс языков, распознаваемых недетерминированными машинами с полиномиальной ёмкостью*.

В отличие от классов по временной сложности, вопрос о соотношении классов PSPACE и NPSPACE решается достаточно легко.

ТЕОРЕМА 11. PSPACE = NPSPACE.

Доказательство этой теоремы можно найти в [2, §10.6].

Из теоремы 2 следует, что для любой функции f справедливы вложения $\mathrm{DTIME}(f(n)) \subseteq \mathrm{DSPACE}(f(n))$ и $\mathrm{NTIME}(f(n)) \subseteq \mathrm{NSPACE}(f(n))$. Следовательно, $\mathrm{NP} \subseteq \mathrm{NPSPACE}$. Поэтому соотношение рассмотренных классов можно изобразить в виде следующей схемы.



ЛИТЕРАТУРА

1. Гэри М., Джонсон Д. Вычислительные машины и труднорешаемые задачи. М.: Мир, 1982. — 416 с.
2. Ахо А., Хопкрофт Дж., Ульман Дж. Постороение и анализ вычислительных алгоритмов. М.: Мир, 1979. — 535 с.
3. Компьютер и задачи выбора. М.: Наука, 1989. — 208 с.
4. Разборов А.А. О сложности вычислений — Математическое просвещение — сер. 3, вып. 3, 1991 г.
   http://www.mi.ras.ru/~razborov/lecture.ps
5. Goldreich O. Introduction to Complexity Theory – Lecture Notes. http://www.wisdom.weizmann.ac.il/~oded/cc.html
6. Rudich S. Scribe Notes on Computational Complexity Theory:
www-2.cs.cmu.edu/~rudich/complexity/handouts/ScribeNotes/complexityscribenotes.ps
7. Вялый М.Н. «Сложность вычислительных задач» :
   http://www.nature.ru/db/msg.html?mid=1161983&uri=node1.html
8. Кузюрин Н.Н. Курс лекций «Сложность комбинаторных алгоритмов»: http://discopal.ispras.ru/ru.lectures.htm
9. Ерусалимский Я.М. Дискретная математика: Теория, задачи, приложения. М: Вузовская книга, 1998.

Содержание

Введение 3

1 Сложность алгоритмов 3

2 Классы сложности 6

3 Детерминированные алгоритмы 8

3.1 Детерминированная одноленточная машина Тьюринга 8

3.2 Многоленточные машины 9

3.3 Эквивалентность машин 11

3.4 Иерархия классов DTIME 18

4 Недетерминированные алгоритмы 24

4.1 Недетерминированные машины Тьюринга 24

4.2 Классы NTIME, NP и co-NP 26

4.3 Альтернативное определение класса NP 30

5 Классы по емкостной сложности 32

Литература 34